University of Alberta

# Automatic Categorical Data Clustering and Spatial Data Clustering by Consecutive Resolution Refinement

by

**Andrew P.O. Foss**

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2002

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-81393-2

Canada

**University of Alberta**

**Library Release Form**

**Name of Author**: Andrew P.O. Foss

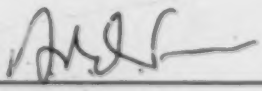
**Title of Thesis**: Automatic Categorical Data Clustering and Spatial Data Clustering by Consecutive Resolution Refinement

**Degree**: Master of Science

**Year this Degree Granted**: 2002

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



Andrew P.O. Foss
202-10610 79 Ave
Edmonton, AB
Canada, T6E 1S1

**Date**: 9/11/02

**University of Alberta**

**Faculty of Graduate Studies and Research**

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Automatic Categorical Data Clustering and Spatial Data Clustering by Consecutive Resolution Refinement** submitted by Andrew P.O. Foss in partial fulfillment of the requirements for the degree of **Master of Science**.

Dr. M. D. Sacchi
External Examiner

Dr. Jörg Sander

Dr. Osmar R. Zaïane
Supervisor

**Date:** 2002 / 8 / 6

Finding a needle in a haystack
simply requires seeing steel and ignoring hay.
- Anonymous

# Abstract

Clustering is the problem of grouping data based on similarity and consists of maximizing the intra-group similarity while minimizing the inter-group similarity. The problem of clustering data sets is also known as unsupervised classification, since no class labels are given. However, all existing clustering algorithms require some parameters to steer the clustering process, such as the famous $k$ for the number of expected clusters, which constitutes a supervision of a sort.

This thesis reviews attempts made to date to resolve the problems in clustering and presents two new, efficient, fast and scalable clustering algorithms free from the need for user input parameters. The first, TURN, is well suited to categorical data while TURN* automatically finds interesting resolution levels in spatial data yielding effective and efficient discovery of arbitrarily shaped clusters in the presence of noise. The experiments show that TURN works well without parameter tuning in comparison to another leading algorithm suited to categorical data while TURN* outperforms most existing clustering algorithms in quality and speed for large data sets.

# Acknowledgements

I would like to thank my supervisor, Dr Osmar Zaïane for his unshakeable support and enthusiasm and constant encouragement. I greatly appreciate and thank Weinan Wang for all the work he did implementing WaveCluster and other algorithms so that they could be compared with TURN and TURN*.

# Dedication

To my mother, Margaret Foss and late father Gr. Cpt. Patrick Foss, OBE

# Contents


**1 Introduction** 1
- 1.1 Problem Statement . . . 6
- 1.2 Contribution Preview . . . 6
- 1.3 Contents Overview . . . 6

**2 Related Work** 8
- 2.1 Partitioning Methods . . . 8
- 2.2 Hierarchical Methods . . . 11
  - 2.2.1 Graph Partitioning . . . 12
  - 2.2.2 Summary . . . 17
- 2.3 Density-based Methods . . . 17
- 2.4 Grid-based Methods . . . 19
- 2.5 Scaling to Many Dimensions . . . 21
- 2.6 Clustering Validation . . . 26
- 2.7 Web Log Mining . . . 29

**3 TURN** 31
- 3.1 TURN: The Method and Rationale . . . 31
  - 3.1.1 Web usage mining . . . 32
  - 3.1.2 What is a Turn? . . . 34
  - 3.1.3 Comparative Analysis . . . 38
- 3.2 TURN Clustering Algorithm . . . 38
- 3.3 TURN: Results . . . 42

**4 TURN*** 47
- 4.1 TURN* Algorithm . . . 47
  - 4.1.1 TURN-RES: Clustering at one resolution . . . 50
  - 4.1.2 TurnCut . . . 52
  - 4.1.3 TURN*: Finding the best clustering . . . 55
  - 4.1.4 Parameter Free? . . . 57
  - 4.1.5 Generalising and Formalising TURN* . . . 58
- 4.2 Experimental Results . . . 61
  - 4.2.1 Clustering Effectiveness Comparison . . . 64
  - 4.2.2 Cluster Efficiency Comparison . . . 68

**5 Conclusion and Future Work** **72**
5.1 Conclusion . . . . . 72
5.2 Future Work . . . . . 73

**Bibliography** **75**

# List of Figures


2.1 CLIQUE succeeds in clustering some points after projecting onto the axes. . . . . 24
2.2 CLIQUE's axes projections lead to a false positive. . . . . 25

3.1 TURN detects cluster boundaries in a time series. From [15]. . 35
3.2 Distribution obtained with Jaccard coefficients and turns found by TURN. From [15]. (negative spikes). . . . . 37
3.3 Visualization of turning points on the third differential. From [15]. . . . . 43

4.1 A: Steps of the TURN-RES Clustering algorithm. B: The general design of *TURN** using TURN-RES to cluster at different resolutions. From [62]. . . . . 47
4.2 A resolution is defined by a distance $d$ along each dimensional axis. At this resolution points A and B are nearest neighbours of point C along the vertical dimensional axis. . . . . 51
4.3 At coarser resolution $d'$ the point D now replaces B as the right nearest neighbour of C along the vertical dimensional axis. . . 52
4.4 At resolution $d$ point A is a close neighbour of C but B is not close: $dist > d$ along the vertical dimensional axis. . . . . 53
4.5 At resolution $d'$ both points A and E are close neighbours of C. 54
4.6 Points are differentiated according to their local density and close neighbourhood to internal points. . . . . 54
4.7 TurnCut finds the optimum clustering resolution for CHAMELEON data set t7.10k.dat. . . . . 56
4.8 *TURN**'s clustering result on t7.10k.dat before cleaning. From [62]. . . . . 64
4.9 *TURN**'s cleaned clustering result on t7.10k.dat after removal of points identified as noise. From [62]. . . . . 64
4.10 K-means's clustering result on t7.10k.dat with $k = 9$. From [62]. 65
4.11 CURE's clustering result on t7.10k.dat with $k = 9$, $\alpha = 0.3$, and number_of_representative_points = 10. From [62]. . . . . 65
4.12 ROCK's clustering result on t7.10k.dat with $\theta = 0.975$ and $k = 1000$. From [62]. . . . . 66
4.13 CHAMELEON's clustering result on t7.10k.dat with $k = 9$ and *nb_closest_neighbor* = 10, MinSize = 2.5%. From [62]. . . . 67

4.14 DBSCAN's clustering result on t7.10k.dat with $\epsilon = 5.9$ and MinPts = 4. From [62]. . . . . 67
4.15 DBSCAN's clustering result on t7.10k.dat with $\epsilon = 5.5$ and MinPts = 4. From [62]. . . . . 68
4.16 WaveCluster's clustering result on t7.10k.dat with *resolution* = 5 and $\tau = 1.5$. From [62]. . . . . 68
4.17 Speed curve of *TURN*$^{\bullet}$ . . . . . 69

# List of Tables


3.1 Clustering Results for TURN. . . . . 45
3.2 Clustering Results for ROCK using a 40% Threshold. . . . . 45
3.3 Clustering Results for ROCK using a 70% Threshold. . . . . 45
3.4 Clustering Results for ROCK using TURNs Threshold Parameters. . . . . 46

4.1 Clustering Speed and Memory Size Results on a data set with 10,000 data points . . . . . 71
4.2 Average Clustering Speed of TURN-RES on one resolution with different data set sizes . . . . . 71

# Chapter 1

# Introduction

Clustering is one of the principal canonical tasks of data mining. Essentially it is the problem of grouping data based on similarity and consists of maximizing the intra-group similarity while minimizing the inter-group similarity. The problem of clustering data sets is also known as unsupervised classification, since no class labels are given.

Clustering is based on the reasonable premise that a data set is heterogeneous and (sic) some items or nodes have a greater affinity to each other than to the rest. In general, the situation is fuzzy, that is the division into groups or clusters is not clear-cut but based on some reasonable assessment of relative affinities. The clustering process has to optimize this similarity between the clustered members vis-à-vis the dissimilarity between clusters and thereby reveal information about the dataset. This allows a kind of data compression which permits us to process the data in a more simple and efficient way and thereby make the necessary decisions.

Rather than holding data about $N$ points or nodes only $N'$ are held where $N' << N$ and each of the $N'$ acts as the definition for all its members. An example of the utility of this would be processing a raster image and converting it into a vector one where each shape in the image had been identified and captured as a whole making it easy to manipulate, or the identification of groups within a customer database making it possible to provide them with information likely to be of interest to them as a group.

Successful clustering should mean that one represents the data set in a

highly compressed way without loosing key features or adding false ones. Points should not be misclassified. Compression comes in two types - loss-less and lossy. Loss-less means finding a representation that is entirely equivalent to the original but with a reduced description length.

However, this typically does not provide the advantages we seek, namely to reduce the potentially very large number of points to a manageable number of classes. In any real world data set there exists randomness, noise, that it would be useful to exclude or segregate. In this and in representing many points by a single class with certain attributes, we choose to loose information as it actually increases our ability to understand and use the data. We could call this useful lossy compression.

Different clustering applications have different requirements regarding the acceptability of information loss. For example, WaveCluster [54] which is based on standard image processing techniques, freely downsamples discarding many data points as it explores different resolutions. This is typically acceptable in image processing but may not be in many data mining applications. The algorithms presented here retain all the points with little cost in processing time ensuring the output is representative of the entire data set.

The clustering problem, also known as unsupervised classification in the machine learning field, is not new and has been addressed by statisticians as well as machine learning and database researchers for many years. However, many of the proposed algorithms do not scale well with today's very large data sets, and new efficient solutions are always in demand. Moreover, almost all existing clustering algorithms require significant domain knowledge to determine and tune various input parameters. Determining adequate input parameters is not only an intricate task but is also very time consuming, necessitating numerous iterations to attain acceptable clustering.

Clustering partitions a data space into meaningful groups of data and thus can play a significant role for other data mining tasks, such as determining class labels for a forthcoming classification task. Such an approach has been used in speech synthesis [44]. There are myriad applications that require clustering for the characterization of data distribution. Many have used clustering in

marketing applications for characterizing groups of customers with similar attributes or behaving similarly.

Clustering World-Wide Web access sessions is a typical application in e-commerce or on-line learning as described herein. Discovering groups of related documents, such as collections of similar web documents, is also a common use of clustering [64]. There are also numerous applications related to spatial data, the groupings of genes and proteins, etc. Image processing is a major area where clustering of 2D spatial data is important, the identification and separation of objects and removal of noise being primary tasks.

There exist a multitude of algorithms for clustering data. An interesting survey [22] covers the most typical and recent approaches. Basically, they each try to concentrate on some important issues in clustering, such as high dimensionality problems, efficiency, scalability with data size, sensitivity to noise in data, identification of clusters with various cluster shapes, etc. However, none has managed to take all these factors into account at once.

The major drawbacks of existing clustering algorithms include the splitting of large genuine clusters, which is the case for partitional approaches such as k-Means [30] and CLARANS [49]; failure to consider concave and elongated shapes of clusters by assuming convex spherical shapes, which is the case of most hierarchical approaches such as BIRCH [65], CURE [17] and ROCK [18]; and the sensitivity to noise in the data (k-Means) or the inability to handle it as in CHAMELEON's case [38].

The greatest difficulty in the field of data clustering is the need for input parameters. Many algorithms, especially the hierarchical methods [21, 65, 17], require the initial choice of the number of clusters to find. Even where this is not required or the algorithm can stop automatically before that number is reached, other parameters greatly influence the output. For example, DBSCAN [10] does not require the input of $k$, the number of clusters, but necessitates the input of two other parameters: the radius of a density reachable neighbourhood and a minimum number of points in the neighbourhood to define a density reachability between points.

ROCK, which is capable of discontinuing the clustering before reaching $k$,

requires the definition of an extremely sensitive threshold for the maximum inter-connectivity between clusters to merge. Since the user, whether human or other software, generally does not know the number of clusters to be expected or what other parameters will be best in advance, and as data can be multidimensional which is difficult or impossible to visualize, this is a serious obstacle. Other challenges are related to dealing with very large databases and high dimensionality. A good clustering algorithm should be scalable to many dimensions and very large data sets. These are two different problems. Scaling to large datasets is covered in this thesis (see Chapter 4) while scaling to many dimensions will be presented in later work.

Other issues are related to the generality of the algorithm. Many algorithms are only applicable to certain types of clusters. A primary example is dealing with arbitrary cluster shapes. For example, one cluster may be contained within another and may, in fact, inhabit the centre of gravity of the surrounding cluster. This defeats k-means and many other algorithms. It is interesting to note that many academics still feel that clusters are inherently spherical and thus k-means remains popular.

Another issue is handling clusters with different densities.Looking at a map of a metropolitan area shows varying population densities typically increasing towards the centre. A good clustering algorithm should tell us that the city as a whole is a cluster and tell us the sub-clusters within it defined by detectable density level changes, such as the downtown area. Some of the existing clustering algorithms might be adaptable to do this but this has not yet been done. It is certainly a highly desirable type of result and will be addressed in future work. However, here methods are presented that capture all the information required to accomplish this. Finally, the time and memory space complexity should be minimised avoiding where possible polynomial complexities or higher.

Another very important consideration is the different types of data that may have to be clustered. While much data is numerical, a great deal is categorical. Categorical data has no trivial Euclidean representation creating two challenges. Firstly, a definition of distance has to be defined which is

usually a similarity measure. Secondly, a location of a point can not in general be defined so even though one can find which points are similar one can not necessarily project the data set onto a low dimensional Euclidean space. One may not be able to define any type of metric space.

Examples of categorical data clustering are document classification and the grouping of web access sessions in data mining from large web access logs. Web logs represent a vast resource of information but one that can only be tapped through the techniques of data mining. It can reveal usage patterns for the benefit of site managers such as the educators whose logs are studied in this work. The results of clustering can give insight into the users' behaviour in a web site and have significant applications in personalization, recommendation systems, adaptive sites, etc.

Only a few clustering algorithms are applicable to categorical data and, like ROCK, they require input parameters. Categorical data is difficult to visualize just because it cannot be represented in a metric space and thus evaluating the clustering and determining correct parameters is non-trivial. In this thesis, a new algorithm that does not require input parameters is presented.

In the particular application studied, an intelligent web-based learning environment for distance education, a major user is the educator trying to evaluate the learning process of on-line students based on access history from web logs. This specific user is not necessarily savvy in data mining techniques, and thus requesting parameters to oversee and control the mining process would be cumbersome and even objectionable. Very few reports on such real web access log clustering exist and the methods used all require input parameters.

In this thesis it is asserted that the most natural way of determining whether data point A lies in cluster X or cluster Y is by determining the natural boundary between X and Y. If that is found, then the data point can be easily assigned to the appropriate cluster. If X and Y are areas of high density of data points, then somewhere between them the data point density will decline to a minimum and start rising again. This is the natural cluster boundary and is an example of a turning point or minimum in the distribution. Since, locally, there can be many turning points, the main challenge is to rank

the turning points. Examples of major and minor turning points discovered by the algorithm TURN are seen in Figure 3.1 in Chapter 3.

This approach of finding the minima is developed in two different ways in the TURN* algorithm for spatial/numerical data (Chapter 4). One method taking advantage of the metric domain is used for finding the cluster boundaries and clustering at a given resolution $r$, then the same technique used to cluster categorical data is used to find the interesting resolution levels.

## 1.1 Problem Statement

It is proposed that it is possible to cluster both numerical and categorical data in an automatic way without the need for any user defined parameters and that this solution can scale to handle very large data sets.

## 1.2 Contribution Preview

This thesis presents new original algorithms which allow for fast efficient automatic or semi-automatic clustering of very large datasets, that are capable of discovering clusters of arbitrary shape, find clusters at different resolutions or densities and scale nearly linearly with dataset size.

Clustering algorithms are presented that handle both categorical and spatial data and in both cases operate without parameter input. The experiments presented show that these algorithms outperform most existing clustering algorithms in quality and speed for large data sets.

## 1.3 Contents Overview

This section has provided an introduction to the challenges being addressed, stated the problem and given a glimpse of the contribution being presented. In the next section, related work will be discussed. After that, the methods and experiments will be presented in two chapters. The first covers the TURN algorithm and its application to clustering categorical data where an Euclidean space can not be defined. In the next, the algorithms of the TURN* approach

for spatial data are outlined and the results obtained presented. In the final section, we conclude and discuss the work that will be pursued in the future to make this a complete clustering solution for higher dimensional spaces with variable densities.

# Chapter 2

# Related Work

There are primarily four groups of clustering methods: partitioning methods, hierarchical methods, density-based methods and grid-based methods. We give a brief introduction to these existing methods in this section.

## 2.1 Partitioning Methods

Supposing there are $n$ objects in the original data set, partitioning methods break the original data set into $k$ partitions. The basic idea of partitioning is very intuitive, and the process of partitioning is typically to achieve certain optimal criterion iteratively.

The most classical and popular partitioning methods are k-means [31] and its derivatives such as k-medoids [40] and k-modes [29]. In k-means each cluster is represented by the gravity centre (GC) of the cluster. The data is clustered into the $k$ partitions that minimise a square-error function, where $C_i$ is cluster $i$ and $m_i$ is it's mean (GC) which is not an actual data point but an abstract location:

$$E = \Sigma_{i=1}^{k} \Sigma_{x \in C_i} \|x - m_i\|^2$$

K-means proceeds as follows. $k$ points or nodes are selected at random. All other points are assigned to these according to nearness. Then a new GC is computed for each cluster and the process is repeated until the termination criterion function given above converges. This will often mean that the process terminates at a local optimum. K-means is quite efficient with a complexity

of $O(ktN)$ where $t$ is the number of iterations, $k$ is the number of partitions and $N$ is the number of data points. Usually $k, t << N$. However, k-means is extremely sensitive to noise. To counter this, k-mediods replaces the GC by one of the "central" real objects of the cluster. A version of this, PAM [40], selects $k$ representatives at random and then for each other point a total swapping cost is computed for replacing any of the selected points with this point. If the clustering is improved by performing the swap, this is done and this process is iterated until no improvement occurs. This approach does not scale to large data sets so the authors created CLARA [40] which draws multiple samples of the data set, applies PAM on each sample, and gives the best clustering as the output. This deals with larger data sets than PAM but efficiency depends on the sample size and a good clustering based on samples will not necessarily represent a good clustering of the whole data set if the sample is biased.

CLARANS [50] is an improved k-medoid algorithm. It does not work with a set of fixed samples but draws a sample with some randomness at each stage of the process. CLARANS searches for a local optimum and then having found it starts again with a new initial sample. CLARANS has been shown to outperform PAM and CLARA and it can also be used to find the most "natural" $k$ and detect outliers but it has a complexity of $O(N^2)$. CLARANS assumes that the entire original data set can be held in the main memory which may not be true for huge data sets. All these partitioning algorithms basically differ in the way they choose cluster representatives in the iterative process.

All the partitioning methods have a similar clustering quality, and the major difficulties with these methods include:

- The number of clusters to be found $k$ needs to be known prior to clustering requiring at least some domain knowledge which is often not available;
- it is difficult to identify clusters with large variations in sizes (large genuine clusters tend to be split);

- the method is only suitable for concave clusters.

A new development on the standard k-means algorithm is bisecting k-means [58]. This has been developed for clustering documents. For any cluster of documents a centroid is defined as the normalised sum of the document vectors in the term-space (set of document words) after removal of common words and stemming.

The first difference from basic k-means is that centroids are updated incrementally, as each point is added, rather than at the end of a pass over the entire data set. The algorithm proceeds by selecting the largest cluster and splitting it into two using basic k-means. This iterates until the desired number of clusters is reached. In experiments on a number of documents, bisecting k-means produced lower entropy results than k-means and several hierarchical clustering algorithms. By the nature of the algorithm, bisecting k-means tends to produce clusters of similar sizes unlike k-means, which tends to result in lower entropy as large clusters will often have higher entropy. If the "correct" result has clusters of different sizes then bisecting k-means will fail. However, the authors' research suggests that this is a fast and efficient approach to document clustering with a computational complexity of $O(N)$ that can also efficiently produce document hierarchies. The authors' also showed that hierarchical algorithms can be improved by applying k-means to their final result.

K-modes [29] is a very straightforward extension of k-means for categorical data. It uses a simple matching dissimilarity measure defined as the total mismatches of the corresponding attribute of two objects. It defines a mode of a set of objects $X = x_1, x_2, x_N$ as a vector $Q = [q_1, q_2, q_m]$ that minimises

$$D(X, Q) = \sum_{i=1}^{N} d(x_i, Q)$$

where $d(x_i, Q)$ is the dissimilarity between $x_i$ and $Q$. K-modes also favours spherical clusters.

## 2.2 Hierarchical Methods

A hierarchical clustering algorithm produces a dendrogram representing the nested grouping relationship among objects. In the past few years, many new hierarchical algorithms have been published. Most are agglomerative, assuming all objects are initial clusters and merging similar clusters until reaching $k$, while some are divisive, initially considering all objects in one cluster and dividing clusters in dissimilar groups until reaching $k$ or another stopping condition. The major difference between all these hierarchical algorithms is the measure of similarity between each pair of clusters and the underlying modelling of the clusters. Because these algorithms are typically computationally expensive, many proceed by sampling the data and clustering only a representative sample of the data points, which puts the effectiveness of the clustering at the mercy of the goodness of the sampling method.

BIRCH [65] introduced the concept of clustering feature and CF-tree, a data structure that summarizes the statistical characteristics of the data points and supposedly preserves the inherent clustering structure of the data. A first pass through the database allows the construction of the CF-tree, then any clustering algorithm can be used on the leaf nodes to partition the objects hierarchically. BIRCH is particularly suitable for large data sets, as long as the CF-tree can fit in main memory. However, the implementations published BIRCH does not perform well with non-spherical shaped clusters, or clusters with large sizes. In other words, BIRCH made breakthroughs on the efficiency issue, but not on the effectiveness issue of clustering.

Instead of using a single point to represent a cluster in centroid/medoid based methods, CURE [17] uses a constant number of representative points to represent a cluster. The constant number of representative points of each cluster are selected so that they are well scattered and then "shrunk" towards the centroid of the cluster according to a shrinking factor $\alpha$. The shrinkage is performed in such a way that this set of representative points keeps the shape and size information of the cluster. The similarity between two clusters is measured by the similarity of the closest pair of the representative points

belonging to different clusters. With proper parameter selection, CURE remedies the problem of favouring clusters with spherical shape and similar sizes, and it seems that it is not sensitive to outliers, even though some false outliers can sometimes be found within legitimate clusters (See Figure 4.11 in Chapter 4). We found that CURE still had a strong tendency to find spherical clusters and could not handle arbitrary shapes. Further, the problem of CURE is its global similarity measure which makes it ineffective with complex data distributions.

Another interesting method by the same authors as CURE is ROCK [18]. ROCK operates on a derived similarity graph. Consequently, ROCK is not only suitable for numerical data, but also applicable to categorical data. ROCK's authors argue that the similarity between each pair of clusters can be measured by the normalized number of total links between two clusters. This normalization of the total number of links is based on a fixed global parameter ($\theta$). ROCK is extremely sensitive to this threshold $\theta$. This fixed parameter actually reflects a fixed modeling of clusters based on a global view of cluster density.

ROCK's clustering result is poor for complex clusters with various data densities, and the algorithm is sensitive to noise. It took much adjusting of parameters to get ROCK to do fairly well with a test data set with 9 obvious clusters. Putting 9 as the target number of clusters produced one large cluster and 8 small noise ones. The best result came from putting $k = 1000$ which caused ROCK to find 5 large clusters and 995 small (noise) ones as can be seen in Figure 4.12 in Chapter 4.

### 2.2.1 Graph Partitioning

Another approach to hierarchical partitioning is to represent the data set as a graph and apply known graph partitioning algorithms. METIS and hMETIS [35, 37] are very efficient graph partitioning algorithms for graphs and hypergraphs respectively. The problem of computing an optimal bisection of a graph is NP-complete but many heuristic algorithms have been developed. Effective multilevel partitioning techniques [5, 26, 25, 36] consist of three phases, coars-

ening, initial partitioning and uncoarsening and refinement. During the first phase, successively smaller graphs are constructed. The coarsest (smallest) graph is then bisected, and then this bisection is successively projected to the increasingly finer graphs, while at each level an iterative refinement algorithm is used to improve the bisection.

An example is Associate Rule Hypergraph Partitioning [47] whose authors report success with document clustering but make no mention of how they terminated the bisecting process. Graph partitioning needs a termination condition as well as certain heuristics to determine the cuts made.

Another example of graph partitioning is CHAMELEON [38]. This builds a k-nearest neighbour graph and partitions it into many small clusters based on the edge weights which represent similarities. It uses hMETIS to perform min-cut bisections on the largest sub-cluster existing at that stage, subject to the constraint that each of the sections found $C_i^A$ and $C_i^B$ contain at least 25% of the original number of nodes in the cluster $C_i$ bisected. This is often referred to as a balance constraint and hMETIS is effective within such a constraint. However, this constraint can cause a natural cluster to be broken.

This process continues until the larger sub-cluster contains less than MINSIZE nodes. This phase has a complexity of $O(Nlog(N))$ while the next phase (cluster merging) is $O(N^2)$. A larger MINSIZE speeds up CHAMELEON reducing the number of clusters to merge in the second phase but if too large, clustering precision suffers. Similarly, the choice of $k$ is a trade-off. The algorithm speeds up with smaller $k$ but the graph can become too sparse and the process of merging may stop short as relative connectivity goes to infinity.

CHAMELEON then proceeds to merge small clusters at each step taking the pair(s) that are most similar. This is based both on relative interconnectivity and relative closeness. It has two schemes, one merges all pairs where these two measures are above some user specified threshold and the second, merging the one pair with the highest closeness based on combining the two measures of closeness using another user specifiable parameter $\alpha$ to weight the combination.

CHAMELEON offers these two measures as an advance on previous meth-

ods that use only one. It is quite effective in clustering various shapes and densities but cannot handle outliers, simply including noise into the nearest cluster. Computing both these measures is fairly expensive so the algorithm is slow to handle large databases and the many parameters involved in the algorithm is a major drawback. CHAMELEON's comparative performance is investigated in this thesis.

Another type of graph partitioning approach is AUTOCLUST [13, 12]. AUTOCLUST claims to be a parameter free clustering algorithm that can detect clusters with differing densities at the same time, i.e. distinguish between them. It uses Delauney Diagrams which the authors state are superior to Delauney triangulations as they 'remove ambiguities from Delauney triangulations when co-circular quadripoles are present'.

A Delauney Triangulation (DT) is an efficient triangulation of a set of points within the convex hull (the smallest path connecting all the outermost points) which gives the most equilateral triangles. This is accomplished by disallowing any fourth point to fall within the circumcircle of any three other points. More formally,

- A Delaunay diagram of a set S of points consists of the segments that belong to all Delaunay triangulations of S.
- A Delaunay diagram is a subgraph of every Delaunay triangulation.
- The Delaunay diagram is a planar graph whose bounded faces are convex polygons all of whose vertices are co-circular.
- If no four points of S are co-circular then all bounded faces are triangles and the Delaunay diagram is a triangulation.
- E.g. see (*http : //www.algorithmic − solutions.com/leda_guide/geo_algs/delaunay_diagrams.html*)

Delaunay triangulations are in general not unique. It is not clear that the AUTOCLUST paper is correct in their distinction. However, they have employed, in fact, Delaunay triangulations which are a duality of Voronoi

diagrams and have already been employed in clustering [9, 33, 11] as the edge lengths provide a way of defining local density. If a triangle contains a short edge and two longer edges, it is likely that the point at the end of the longer edges is in a different cluster. The AUTOCLUST authors claim that these earlier methods failed to eliminate co-circularity and thus returned inconsistent results. They may mean that triangulations constructed at different times were different as a DT is not necessarily unique, even though each triangulation, by definition, excludes co-circularity. More seriously, the earlier methods used global thresholds to decide when to remove an edge to separate clusters and this inevitably means the results are parameter sensitive and fail to handle differing cluster densities. However, these methods were still quite successful showing the value of the DT.

Automated clustering means that the user is not required to input any parameter settings. Since crisp clustering involves decisions - point $P$ is/is not in cluster $C$, there is always a threshold value(s) involved. Successful auto-clustering approaches such as AUTOCLUST and TURN*, the new algorithm presented in this thesis, find thresholds that are robust across differing data sets due to their inherent 'sensibleness'. While earlier papers using DTs simply fixed a parameter value, the AUTOCLUST authors resort to an equivalent method to TURN* in that they attempt to define 'internal' vs. 'boundary' points and they accomplish this by using local and global means and standard deviations (SD) of the edge length.

A local mean (LM) for point $p$ is the mean length of all the $n$ edges incident on $p$. From these values a local SD is computed. However, since $n$ is small, a global SD value (MSD) is computed as the mean sum over all local SDs. Then all edges are divided into short: $length < LM - MSD$ and long: $length > LM + MSD$ and all other edges.

An edge, since it links two points, can be classified into two categories. Clearly long edges can be assumed to be between different clusters. Short edges may be on "noise bridges" as the bridge edges are very short given the long nature of the other edges coming into the point (from noise, for example). Short edges are also removed and some have to be added back in later in case

they are determined to be inside a cluster.

After this first step of removing long and short edges there are two more "clean up" phases. The first uses the short edges to choose where "other edges" connect to different candidate clusters. In the second, a $k$-neighbourhood is defined, that is the set of $k$ or less edges starting at a point $p$. A local mean is computed for these $k$ edges and long edges removed. Here a second parameter has been introduced $k$, the first being embodied in the implied constants in the equations above defining "long", etc. edges. The AUTOCLUST authors take $k = 2$ but do not defend their parameter choices. However, they do demonstrate the success of their choices on the data sets tested.

While 2D DT can be constructed with $O(Nlog(N))$ complexity, this does not scale to higher dimensions. Some work has been done building Delauney spheres being dual to the Voronoi polyhedra (developed by Voronoi in 1908) for genetic work but this is only 3D and the complexity can go to $O(N^2)$ (e.g. [60]).

The other common way of handling a sparse graph, other than DT is $k$-nearest neighbour. This is used by Harel and Koren [24, 23] to cluster spatial data using random walks. This has the same scaling problems for higher dimensions as all other approaches reviewed here as the spatial index structures break down. The paper argues for using the intersection of DT and $k$-nearest neighbour which can be computed given both in $(O(N))$ time, however, the difference from the $k$-nearest neighbour graph is often slight. As with all clustering, the challenge is to find where there is a sharp local transition of some characteristic of the cluster indicating a potential boundary. This is why algorithms that take a more local view usually do better than those which use global values.

These authors attempt to sharpen edge weights so as to increase the "internal"/"external" edge difference using deterministic analysis of random walks. An example of how the authors use random walks is by comparing the likelihood of a walk that visits point $v$ reaching point $u$ before returning to $v$. Clearly if $u$ and $v$ are remote, this is low and then the points are treated as likely to be in different clusters. The approach assigns weights to the edges

between points/nodes and 'sharpens' the differences by adjusting the weights repeatedly until the "intercluster" edges are highly differentiated from the "intracluster" ones.

### 2.2.2 Summary

A common disadvantage of hierarchical clustering algorithms is setting the termination condition which requires some domain knowledge for parameter setting. Further parameters have to be set and generally the computational complexity is high $O(N^2)$.

## 2.3 Density-based Methods

Density-based methods identify clusters through the data point density and can usually discover clusters with arbitrary shapes without a pre-set number of clusters.

DBSCAN [10] is a very successful density-based method that connects regions with sufficiently high density into clusters. Each cluster is a maximum set of density-connected points. Points are connected when they are density-reachable from one neighbourhood to the other. A neighbourhood is a circle of radius $\epsilon$ and reachability is defined based on a minimum number of points $MinPts$ contained in the radius $\epsilon$.

DBSCAN, however, is rather sensitive to the selection of $\epsilon$ and $MinPts$. Moreover, it cannot identify clusters with different densities. The problem of discovering clusters with different densities and clusters within clusters was alleviated in OPTICS [42] by some of the same authors. OPTICS is a extension to DBSCAN that eases the pre-setting of the radius ($\epsilon$) by producing an augmented ordering of the database representing its density-based clustering structure. This cluster-ordering contains the information about every clustering level of the data set (up to a "generating distance"). However, in OPTICS the parameter $MinPts$ still needs to be defined.

With proper parameter setting, DBSCAN and OPTICS are experimentally very effective for spatial data clustering. Other density-based clustering algo-

rithms include DENCLUE (DENsity-based CLUstering) that clusters based on density distribution functions. DENCLUE models the overall point density analytically as the sum of influence functions of the data points with the clusters being identified by determining density attractors. While DENCLUE was introduced specifically for high-dimensional data, it is based on kernel density estimation and it has been shown that a density estimate based on an arbitrary kernel becomes insignificant when the dimensionality of the data grows [8, 6]. Thus such algorithms are not able to deal with the inherent sparcity of high dimensional feature spaces.

DENCLUE provides a description of a data set in the form of a curved space that can be cut at different densities using their parameter $\xi$ to provide different resolution clustering results in much the same way as OPTICS potentially could. In fact $\xi$ is closely equivalent to the parameter MinPts in DBCSAN but, here, is a lower bound on the density function value of a density maximum or "density-attractor" in the authors' terminology. However, the authors don't capitalize on the potential for assessing different resolutions. They focus on the use of $\xi$ to speed computational time and filter out noise based on their assumption of uniform noise distribution.

In real applications noise is far from uniform - it is clearly a question of defining noise. Like a weed in a garden is simply a plant you don't want, noise is either a white noise background, as they assume, or data points with any kind of distribution that have no relevance to the knowledge that we want to discover. Generally, noise is simply assumed to be sparse compared to useful data clusters.

The DENCLUE authors propose a heuristic for finding the optimum value of their parameter $\sigma$, that determines the influence of a point in its neighbourhood, by picking a value in the longest range over which the number of density attractors $m(\sigma)$ is constant. $\sigma$ is related to the parameter $\epsilon$ in DBSCAN and controls the shape of the Gaussian influence functions used to fit the distribution. The Gaussians are defined in the usual way as:

$$f_{Gauss}(x, y) = e^{-d(x,y)^2/2\sigma^2}$$

This heuristic is a rather simple approach to the problem of finding the "optimal" clustering resolution, which is addressed in this thesis in Chapter 4. Another point of interest is their concept of highly populated cubes and populated cubes. First they divide the space into populated cubes $C_p$ and then select from them those with a high density of points $C_{sp}$ based on a parameter. Clustering proceeds with only $C_{sp}$ and the $C_p$ connected to them considered.

The DENCLUE authors state that the time to access the cubes for an arbitrary point is $O(log(C_p))$ however this relies on relatively low dimensionality. Logarithmic access tends to break down for $D > 16$ as the trees used to index the locations of data nodes loose their performance advantage. The paper gives results for $D = 11$ and they state that they ran tests successfully on $D = 19$ without giving any numerical data on the performance.

## 2.4 Grid-based Methods

Grid-based methods first discretize the clustering space into a finite number of cells, and perform clustering on the gridded cells. The main advantage of grid-based methods is that the processing speed only depends on the resolution of griding and not on the size of the data set. The grid-based methods are more suitable for high density data sets with huge number of data objects in limited space.

Representative grid-based algorithms include STING [61], CLIQUE [2] and WaveCluster [54]. STING builds a description of the data set by allocating all points to cells of the imposed grid and maintains certain statistical values for each cell such as *count*, *min*, *max*, *distributiontype*, *mean* and *standarddeviation*. The distribution type is either assumed by the user or obtained from a set of hypotheses using, for example, a $\chi^2$ test.

A multilevel approach involves building coarser layers of cells and their data from the finest level. These levels can be queried to extract information at any required level or becoming increasingly finer until the query specification is met. Of course, the data will usually not be in the hypothesized distribution and the imposition of a grid discards information about cluster boundaries that

goes finer than the finest level of the cells. On the plus side, building such a database involves one pass over the data set and querying the data is also fast.

The recent clustering method WaveCluster first summarizes the data by applying a grid structure on the data space, followed by a wavelet transformation. The grid-based quantization of the data space speeds up the processing, but due to the rectangular structure of all grid based approaches, the algorithm represents a much coarser approach to levels of resolution than that adopted by the algorithm *TURN** presented herein, or DBSCAN, and is much more likely to lead to the misclassification of data points. The algorithm requires this quantization so one does not have the option of classifying the data at full resolution. The data is subjected to high and low pass filters and downsampled by two and this is repeated in each dimension $D$. The result is $2^D$ signal components of which the output from using only the low pass filter ($LL$) is actually used.

The WaveCluster approach is essentially one of applying a filter to the data resulting in solid "dark" areas and "grey" or "white" areas where noise has been suppressed. Then the dark areas are stitched together by assigning the same cluster number to all neighbouring "dark" areas - "dark" is differentiated from "grey" or "white" by a threshold parameter. WaveCluster offers multiresolution by skipping "rows" of the quantized data (i.e. down sampling), again a much cruder approach than that used by the other algorithms that use all the data and simply expand the nearest neighbour definition or equivalent of each point. WaveCluster's main benefits are speed and scalability since the data is read in and quantized and then subsequent processing is on a much smaller effective data size.

The wavelet transformation is of a very simple type, essentially differencing the values between neighbouring cells along dimensional axes. This has certain parallels with an approach taken in this thesis (see Chapters 3 and 4) but on a coarser level (due to the grid approach) and considers only the density along that axis as opposed to the TURN* approach which keeps a value of the local density in all dimensions. The failure of WaveCluster to cluster the test data presented here was a direct result of this even though we manipulated the

parameters to try to find a perfect solution. A noise bridge that happened to be dense (while narrow) along a vertical axis defeated it (see Figure 4.16 in Chapter 4).

While WaveCluster claims to be parameter free, it has two key parameters $\tau$, the signal threshold for deciding whether a cell is a significant cell in $LL$ (a density threshold), and the resolution. Our research showed that its clustering results are quite sensitive to the settings that have to be made. In fact, the paper [54] points out that knowing the number of clusters to be found is very helpful for choosing the parameters for WaveCluster.

## 2.5 Scaling to Many Dimensions

As the number of dimensions $D$ of a data set increases, data gets sparser and any clustering effect is reduced. One can understand this intuitively but it has also been shown [4] theoretically. Furthermore, while there are low dimensionality $O(Nlog(N))$ strategies for finding the nearest neighbours of a node, for $D > 16$, strategies, such as indexed trees (e.g. SR-tree), fail and the computational complexity goes to $O(N^2)$. Indeed, [4] showed that linear scan beats indexing strategies in almost all cases for $D = 10$ or higher.

The authors also showed that indexing strategies could still be useful in certain cases, e.g. where clusters are tight and well defined and queries can be guaranteed to fall in or very close to a cluster and where dimensionality can be reduced - i.e. the clusters exist in a much lower dimensional space. In the method presented in this thesis, where dimensions are sorted, one may be limited by the length of each point's description as sorting is ideally performed in main memory, though there is only a time cost in handling larger datasets using disk space.

In this context it is interesting to note that Grid based approaches are equivalent to compressing the description length. Instead of holding $n$ neighbouring points with exact locations/attributes, the description length is compressed until all $n$ descriptions are the same and then the $n$ descriptions are replaced with a single one with a count added (and optionally other statistics).

A half-way house approach, is to compress the descriptions without removing duplicates. Indeed, to find the duplicates would involve either a sort or an equivalent $O(Nlog(N))$ approach. Those authors (e.g. WaveCluster) who claim an $O(N)$ complexity on the grounds that all the Grid based approach needs is to read in the data once ignore the reality that putting each point into the appropriate bin costs either $O(NN')$ (linear search) or $O(Nlog(N'))$ (indexed search) where $N'$ is the number of bins unless one can hold a matrix for the whole space in memory - in other words they appear to assume this.

However, this would be infeasible for large datasets and also high dimensionality where even indexed searching breaks down. Or they assume that $N' << N$ and therefore $O(NN')$ is not different from $O(N)$. However, even with a very coarse representation, as dimensions increase $N'$ rapidly increases and can vastly exceed $N$. For example, a grid with only 100 units along each dimension and only 20 dimensions already has more cells than most large datasets have points.

Rather than using the grid approach with these inherent difficulties, one could compress the full-dimensional data description of the $N$ points until the data fits in memory or a reasonable proportion does. For example, if one has 10M data points in 100 dimensions each requiring 4B to describe it, the dataset is 4GB. If the available memory is 500MB, compression of 4B to 1B would allow half of the data set to be loaded in main memory. Such techniques have not so far been specifically discussed in the literature in this context.

An alternative approach is dimensional reduction. It may well be that some dimensions contribute little to the clustering information. The choice then is between global dimensional reduction which is fairly straightforward and local reduction which is likely to retain more useful information.

A standard way of dimensional reduction is Singular Value Decomposition (SVD). First the $d \times d$ covariance matrix is constructed. Each entry $(i, j)$ is the covariance between dimensions $i$ and $j$. This is a positive semidefinite matrix. The eigenvectors are found defining an orthonormal system along which the second order correlations in the data are removed. The eigenvectors with the largest eigenvalues are chosen to represent the data as these represent the

variance along each newly defined dimension in the orthonormal system. If the discarded eigenvalues are small then the loss of information is acceptable.

Several authors [59, 34] applied SVD globally. Aggarwal and Yu [1] in ORBCLUS have applied SVD locally. They start by sampling the data and creating $k_c$ clusters using full dimensionality. These are then locally dimension-reduced using SVD. The remaining points are added to the cluster to which each is deemed closest. The number of clusters is reduced (in their case by 50% at each merge). Merging continues until the target number of clusters $k$ is achieved.

Outliers are handled by discarding small clusters at each iteration. This mainly occurs in the early stages. Of course, the chance of discarding an important cluster core is also higher at this stage. The algorithm is impressive but suffers from the old problems of the need for a specified value of k as well as various other parameters, and the dangers of sampling. Since the clustering is most obscured in full-dimensionality and the method is very dependent on the initial clustering done on a sample in full dimensionality, much work would still need to be done to determine how globally effective this algorithm is.

A tempting approach to the high dimensionality problem is to project the data on to a set of axes, as it reduces the problem from, typically, exponential in the number of dimensions to one that is linear in them. We now describe two leading examples of this.

CLIQUE [2] is an algorithm for finding subspace clusters in high dimensions even when the data is very sparse. In fact, it is only useful when the data is very sparse. CLIQUE first seeks high density areas in 1-D projections of the space. The projections are onto a digitised space of $n_D$ units per dimension $D$. It takes an apriori approach, that areas that are dense in $k$ dimensions will be dense in $(k-1)$ dimensions, so finding the dense areas in the lowest dimensional level allows a candidate space to be generated for search. Even then the search space can be very large in high dimensions so a Minimum Description Length pruning method is adopted. Subspaces where the total number of points that fall in the dense units is below a threshold are pruned.

Next they greedily cover the intersecting dense areas of the reduced num-

ber of dimensions and then discard the redundant rectangles. However, this assumes that dense areas in each dimension are related which could easily not be the case. For example, suppose there is a dense unit at $x = 3$ and one at $y = 4$. So it assigns a cluster at (3, 4). However, the cause of the density projected onto the $x$ axis could be due to a string of sparse points at many values of $y$ with $x = 3$ and similarly for the $y$ axis (see Figures 2.1 and 2.2). This could be solved by referring back to the original data but this was not done presumably due to its computational cost.

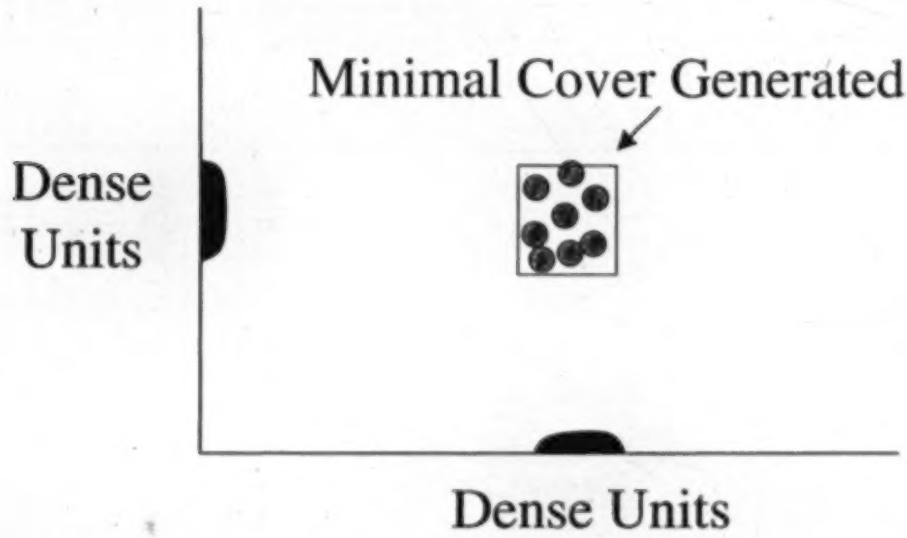



Figure 2.1: CLIQUE succeeds in clustering some points after projecting onto the axes.

The CLIQUE paper contains an interesting discussion of Principal Component Analysis (PCA), that is dimensional reduction using Singular Value Decomposition, which is used in other clustering algorithms discussed here. They show examples where PCA helps and where it serves no useful purpose.

OptiGrid [28, 27] looks very different from TURN*, as presented in this thesis, but in some respects is following a similar approach except that TURN* retains a local approach to finding density minima while OptiGrid projects the data onto each dimensional axis and then looks for the minima in the "compacted" projection. The correct turning points may be obscured by this method. OptiGrid also assumes a "model" or density function approach and

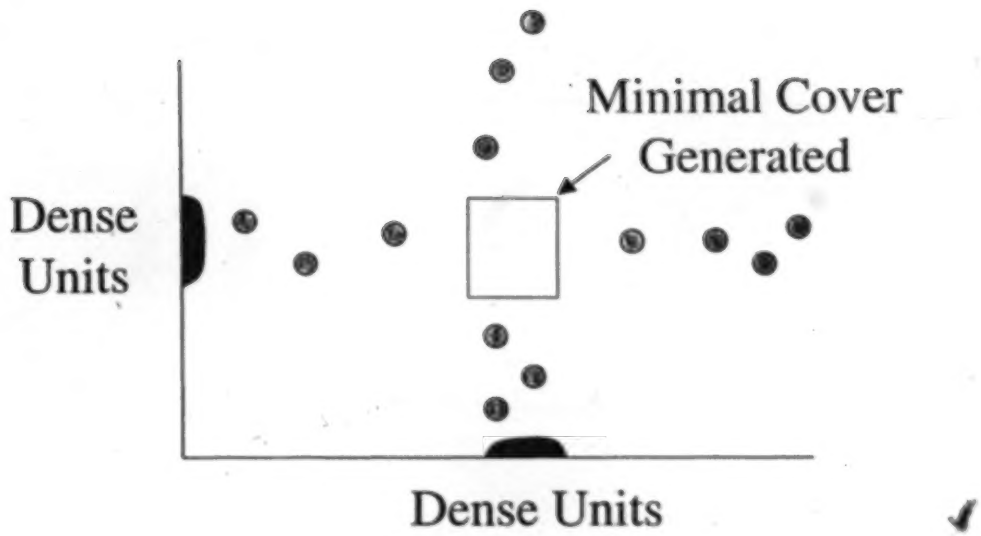



Figure 2.2: CLIQUE's axes projections lead to a false positive.

has only been tested for normally distributed data with noise. However, it proves both efficient and effective for this data in higher dimensionality ($D \leq 20$). The idea is that rather than placing all the data into a fixed grid, an approach that has difficulties at high dimensionality, partition the space with some optimally chosen hyperplanes.

OptiGrid discusses the utility of finding an optimal set of projection axes but actually uses axis parallel projections. The data is projected onto the axes and density minima, below a certain threshold, identified and used for fixing the orthogonal hyperplanes. In this thesis, we make the point that there are generally many minima, of which only a few will be "interesting". Determining which are interesting is highly non-trivial and the approach of OptiGrid hardly addresses this. They simply limit the number of planes to be at most some parameter $q$ out of those "points" that have a "minimal" point density.

Of course, to ascertain the point density typically requires some type of gridding, quantization of the space or k-nearest neighbour definition and the paper does not address how this was done and the effects of this in terms of imposing a kind of resolution on the data. Nor is the problem of handling this given the problems they discuss with regard to higher dimensionality addressed. Just as it is easy to show when CLIQUE fails, a similar exercise can

be done for OptiGrid due to the inherent information loss in axes projections.

## 2.6 Clustering Validation

One of the main difficulties with clustering algorithms is that, after clustering, how can one assess the quality of clusters returned? Many of the popular clustering algorithms are known to perform poorly on many types of data sets. In addition, virtually all current clustering algorithms require their parameters to be tweaked for the best results, but this is impossible if one cannot assess the quality of the output. While 2D spatial data allows for assessment by visual inspection, the result is dependent on the resolution presented to the inspector and most clustering tasks are not 2D or even spatial. One solution is to reduce any output to a 2D spatial presentation. Other solutions are based on 1) external, 2) internal and 3) relative criteria [19].

The early work on 2D representation was done by John Salmon in 1969 [52]. It uses a steepest descent algorithm to minimise the "Sammon stress" which is a measure of the error based on the interpoint distances $d$ given the original interpoint distances $d^*$ in the high-dimensional space. This can be computed in $O(N^2)$ time.

$$sammonstress = (1/\sum_{i<j} d^*_{ij}) \sum_{i<j}^{N} ([d^*_{ij} - d_{ij}]^2 / d^*_{ij}) \tag{2.1}$$

It starts from a random distribution of the points. The high cost can be reduced using a Conjugate Gradient approach (for example). It is efficient in identifying hyper-spherical, hyper-ellipsoidal clustering, whose geometric relationships are obviously governed by the inter-point distances, but one has to start over if any new points added. Also it may stop on local minima but one can make several runs. Another classic approach developed by Kohonen in 1981 is SOM [39]. This uses a Neural Net to map the data onto 2D. It starts with randomly assigned output nodes and then finds the winning (closest) output node for input vector $x$ by comparing it with all $k$ output nodes and then updates the node weights using a neighborhood function. This has value

1 when $i = k$ where k is the winning node and falls off with the distance $|r_k - r_i|$ between units i and k in the output array, thus

$$w_k(new) = w_k(old) + \mu f(i,k)(x - w_k) \tag{2.2}$$

Units close to the winner as well as the winner itself, have their weights updated appreciably while weights associated with far away output nodes do not change significantly. It is here that the topological information is supplied. Nearby units receive similar updates and thus end up responding to nearby input patterns.

More recently, FastMap [14] projects objects and candidate clusters onto a 2D space for visual analysis based on the principle that a cluster that is separated in 2D (or 3D) will be separate in higher dimensions. It uses pivot points for the projection and looks for points as far separated as possible.

Visual Validation methods are intuitive but suffer from certain difficulties:

- Visual assessment is subjective.
- Depends on the resolution presented to the viewer.
- Projection onto a lower dimensional space can always obscure certain structures.

The External and Internal Criteria approaches [19] use Monte Carlo methods to evaluate whether the clustering is significantly different from chance. Certain statistics such as Hubert's $\Gamma$ statistic are computed. Where $M = N(N-1)/2$, this is defined as:

$$\Gamma = (1/M) \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} X(i,j)Y(i,j) \tag{2.3}$$

High values of this statistic indicate a strong similarity between matrices X and Y. Other statistics are based on whether pairs of points agree in their placement in C (the clustering result matrix) and P (the evaluation matrix).

In the External approach, the clustering result C can be compared to an independent partition of the data $P_{int}$ built according to our intuition of the structure of the data set or the proximity matrix P is compared to C. For example, suppose we feel that there are 4 clusters in the data. We can run k-means and set $k = 4$. We can divide the data into 4 partitions randomly. To determine if the resulting statistics indicate a significant difference (i.e. the clustering is producing a non-random result) another 100 synthetic datasets are generated and clustered and the same partitioning defined and statistics generated. This gives us a distribution and hence a $p$ significance value.

The Internal Criteria approach uses some quantities or features inherent in the dataset to evaluate the result. If the clustering is hierarchical, for example, the cophenetic matrix $P_c$ can be created representing the proximity level at which two vectors are found in the cluster for the first time. This is a matrix representation of the dendrogram. A correlation coefficient between $P_c$ and P is computed and this is repeated for many synthetic data sets to determine significance. For non-hierarchichal methods an additional matrix is generated Y where $Y(i,j) = \{1$, if $x_i$ and $x_j$ belong to different clusters and 0 otherwise $\}, i, j - 1, ...., N$. The $\Gamma$ is used to compare Y and P using the same Monte Carlo method to determine significance.

These methods are clearly very expensive in processing time and only tell us that the clustering result is not pure chance. The Relative Criteria does not involve statistical tests but attempts to evaluate which of several results arising from different parameter settings (such as clusters to be found $k$) give the best result. The big challenge is to characterize the clustering result in a way that tells us the quality of the clustering.

Naturally, there is a grey line between measures used by clustering algorithms to determine where to join or split clusters and indices proposed to determine if that was good. Like many clustering algorithms, these indices suffer from problems especially the inability to handle non-spherical clusters. For example, the Dunn index [19] that is related to the single link method and computes the shortest distance between two clusters and divides by the diameter.

Another approach [19] computes several indices such as the Root-Mean-Square Standard Deviation, a measure of homogeneity, and plots them against $k$. Whatever the index, having created a graph, this is inspected visually for either a minima/maxima or a "knee", being the greatest jump of the index with a change in $k$. There is, however, no rigorous way of ensuring that this "knee" identifies the correct $k$. There are different indices defined for evaluating fuzzy clustering. An evaluation done [19] of a number of indices on data that contained only concave but not always circular clusters, found different indices were better on different data sets showing their shape-dependence.

In a few cases, Clustering Validation approaches have been integrated into clustering algorithms giving a relatively automatic clustering process. Smyth [56] presented MCCV, the Monte Carlo Cross-Validation algorithm though this is intended for data sets where a likelihood function such as Gaussian mixture models can be defined. The use of a cross-validation approach permits the use of a log likelihood measure to estimate the posterior probabilities for different values of $k$ clustered using an EM approach but it also makes it computationally expensive.

The TURN* approach, presented in this thesis, contains within it an automated method (TurnCut) for locating the optimum cluster resolutions and represents an implementation of the Clustering Validation approach for automating clustering that handles arbitrary shapes, noise, and very large data sets in a fast and efficient way. Thus it represents a major breakthrough in this field.

## 2.7 Web Log Mining

Web log mining or web usage mining is the application of data mining techniques on web access logs. Most of the current studies in this area are very new, but more and more work is being done. [57] is a recent survey paper which discusses some concept definitions and provides an up-to-date survey of Web Usage mining. [46] described the architecture of the Web Miner system, one of the first systems for Web Usage mining.

WebLogMiner described in [63] uses a multidimensional datacube approach with on-line analytical mining to discover pertinent patterns in web logs but does not perform clustering per se. [32] is a paper defining user sessions and discussing clustering user sessions based on the pair-wise dissimilarities using a robust fuzzy clustering algorithm. There is also interesting work on personalization [48] and web adaptation and evaluation [41] relevant to web usage clustering.

An interesting clustering experiment on web log transactions using the known BIRCH algorithm was reported in [16], but the clustering is performed on generalized transactions using concept hierarchies of pages in a site. That means that diverse URLs that refer to similar concepts are equated, which resolves the problem of very small clusters and identifies users intentions. The paper, in fact, does not go as far as it's method could allow but simply generalizes on a single site using Attribute Oriented Induction [20]. Given an URL A/B/C/D, a two stage generalization reduces it to A/B. The simplified URLs are then clustered. Overall there are few reports on real applications so far.

# Chapter 3

# TURN

## 3.1 TURN: The Method and Rationale

The research presented in this thesis started with an attempt to find a way to suggest parameters for the ROCK algorithm. ROCK is very sensitive to its parameter settings like most algorithms and this makes it nearly impossible to apply it in real-world situations. Having achieved some success with this, it became obvious that the approach could be extended to become a complete clustering algorithm. As the main idea was to find the turning points in the data distribution, the algorithm was named TURN.

As stated in Chapter 1, if X and Y are areas of high density of data points, then somewhere between them the data point density will decline to a minimum and start rising again. This is the natural cluster boundary and is an example of a turning point or minimum in the distribution.

Clustering can consist of two main approaches. One looks for maxima in the data distribution and identifies them as cluster centres and then includes neighbouring points in some way, e.g. DENCLUE as discussed in Chapter 2. The second looks for minima in order to define the cluster boundaries as in OptiGrid reviewed above. the second is compelling because it should allow us to more precisely contain the clusters and avoid failing to find members.

Hence, clustering can be taken as essentially a problem of boundary determination but most clustering algorithms don't attempt to do this directly. The human eye and optic processor actually only sees boundaries, working somewhat like a video games renderer that defines a polygon (the boundary)

and then calls a fast fill routine. Further, given a pattern, a human asked to group the points would look for the spaces between groups. When referring to turning points, we refer only to these minima in the distribution.

The difficulty arises because in any real-world distribution there are many turning points reflecting different levels of resolution. Small variations could be declared noise but one person's noise is another's important data. For instance, the Cosmic Microwave Background was first seen as noise but now we know that its very fine variations reveal the distribution of matter at a very early stage of the universe's history.

Since, locally, there can be many turning points, the main challenge is to rank the turning points. This would then allow us to provide the user with whatever level of resolution she desires and may also allow us to identify certain key levels. Figure 3.1 clearly illustrates the problem, in that there are many maxima and minima in the time series.

In the next chapter, we look at finding cluster boundaries in spatial data and here we pursue an application in a non-metric - categorical space. The application is the analysis of usage of a distance education web site.

### 3.1.1 Web usage mining

Web usage mining allows for the discovery of patterns in the behaviour of visitors to a web site allowing management to optimize the site for the benefit of visitors. Cluster discovery is an important part of finding patterns. The mining usually has three main steps: 1) data preprocessing, 2) data mining, and 3) pattern evaluation. In our work, the data preprocessing step contains two phases: data cleaning, which is done automatically using some heuristics, and data filtering, which is under user control.

Data cleaning involves the removal of entries which contain an error flag, request methods other than GET, requests for images and other embedded files, applets and other script codes, requests generated by web agents such as web crawlers, etc. whose function is to pre-fetch pages for caching, requests from proxies, etc. Some entries are also transformed into 'actions'. For example a CGI script call with given parameters could be replaced with the action

pertaining to the script (i.e. an answer to a quiz question, etc.).

After cleaning the data it is necessary to seek to identify useful groupings of the transactions. Here both users and sessions are identified. Fu et al. [16] describe how these are identified. They employed Attribute Oriented Induction and the clustering algorithm BIRCH [65]. This scaled well over increasingly large data sets and produced meaningful clustering results. However, BIRCH involves the setting of a threshold to determine 'closeness' as well as its sensitivity to the order of data input. [16] does not discuss how they set the threshold or how the 'closeness' between sessions was computed.

In this research, the intention is to avoid user defined thresholds as well as offering additional options to the user beyond but including Attribute Oriented Induction. Mobasher et al. [45] used clustering on a web log using the Cosine coefficient and a threshold of 0.5. No mention is made of the actual clustering algorithm used as the paper is principally on Association Rule mining. The interface developed for TURN offers Cosine [51] as an option. The results with ROCK on the web log investigated suggest that 0.5 may be somewhat low.

Transactions (cleaned web log entries) are grouped into sessions and sessions are grouped, if desired, by user in order to investigate patterns in the usage of the site in individual user sessions or over time by individual users. Before seeking any rules, it is desirable to find if there are any clusters or groupings within this data. We might find, for instance that many of the visitors to a particular university course web page also accessed some help page. This might identify a need to which the administration could respond.

Prior to clustering, we can apply user-controlled filtering. Many filters could be defined. Some obvious ones are 'generalize to level', 'remove duplicate pages', 'remove duplicates to levels', and 'remove short duration pages'. 'Generalize to level' would mean that all URLs would be generalized to a certain level of the site tree, essentially Attribute Oriented Induction [16]. For example, on a university web site, computer science courses might appear on the third level of the university domain - Root/Department/Course. By generalizing to the third level and removing duplicates, one would be able to cluster all sessions or users primarily viewing one course.

Removing short duration pages is a very logical choice to eliminate click-throughs and is generally more effective than Maximal Forward Reference (MFR) [7] for this. Applying MFR was investigated in this study and has very little effect on sites with a strong horizontal movement component. Most web sites today are like this as they contain links on each page to most other pages. However, as a published method that is also referenced by others, it was offered as a filter choice to the user in the interface.

The interface of the application developed for TURN offers the user various ways of filtering the transactions. Here we define a transaction as a cleaned URL in the log file. All filters apply to a single session or user depending on which the user selects to cluster. The user can also choose the similarity coefficient to apply during clustering. Various coefficients have been proposed in the literature such as Jaccard [18], Dice [51], Cosine [51] and Overlap [51].

Jaccard and Dice have been found to be functionally equivalent so we did not consider Dice. We compared Jaccard with Cosine and found little difference so only Jaccard was used for the work presented here. The Overlap coefficient tends to yield a much larger number of identical items, which could lead to very large clusters. For instance, a session consisting of A/B will be judged identical to a session consisting of A/B/C, A/B/C/D, and any number of other URLs. This is likely to give results that a user might judge as extraneous.

### 3.1.2 What is a Turn?

Current clustering algorithms, including WaveCluster, characteristically involve certain heuristics. The approach of this thesis is to detect turning points (minima) without user-defined parameters and without applying any smoothing or noise filters and then apply the approach recursively. To accomplish this, the series in question, which in this case is the difference values from one data object to all other data objects, is repeatedly differenced looking for changes of sign in the resulting values. Such a change of sign is referred to as a "turn".

If one differentiates $d$ times, then $2^d$ points are involved in a turn so the

view becomes increasingly global removing fluctuations that involve smaller numbers of points. Because turning points are used to define clusters we call this approach TURN. It does not depend on any user-defined parameters even in the pre-processing stage.

Figure 3.1 illustrates how TURN picks out cluster boundaries and assigns an amplitude to the change in a distribution in a metric space such as a time series. In this graph, the data is a time series with a couple of events where the measured attribute rose sharply and many minor fluctuations. The bottom graph shows peaks or spikes at the turning points found in the upper graph. The major events in the upper graph are picked out by the largest spikes, which then allow us to group the points in between as clusters. Within this we can see smaller peaks indicating another level of structure and even within the background there are small spikes, which in some experimental situations could represent interesting events.

It can be observed that some of the larger secondary spikes do not indicate sharp changes in the distribution but rather the beginning of a persistent upward or downward trend. The heights of the peaks are used to rank the turns and the algorithm is used recursively to detect clusters within this distribution.

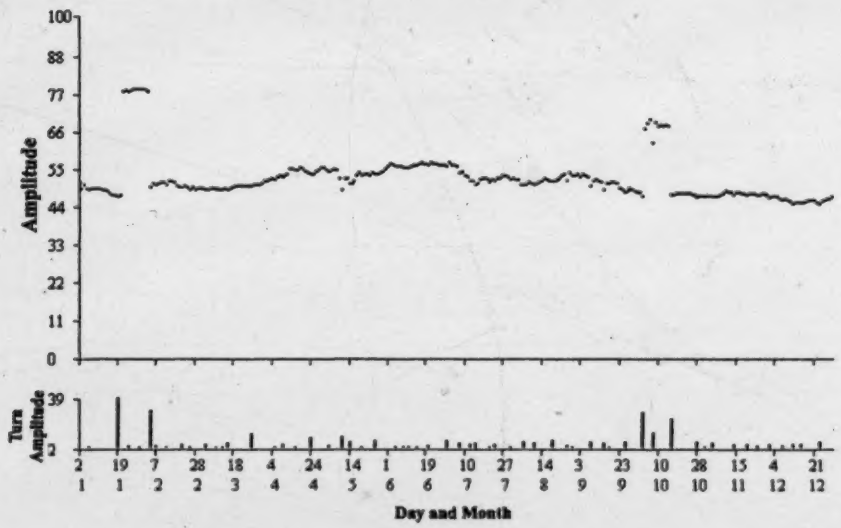



Figure 3.1: TURN detects cluster boundaries in a time series. From [15].

The algorithm presented here is adapted to the problem of a non-Euclidean

space as in the case of web log mining. Web log analysis presents a particular challenge because there is no straightforward way of defining a Euclidean distribution among the data points. Each data point is in fact a graph consisting of many URLs so there is no mean. This precludes the use of clustering algorithms which depend on these such as $k$-means [31], CLARANS [49], etc.

However, one can use measures of similarity to define distance between data items (sessions or [all sessions for] users). If these distances are sorted, one has a series starting from 100% similar and declining to 0% similar. There are no minima but there are turning points in the differentiated series at any level.

In the work reported here $d = 3$ was used, that is the rate of change of the acceleration or second differential of the (similarity, i.e. difference) series. At this level, flattening of the curve after a decline produces a spike or change in sign in the differentiated series. This can be seen in Figure 3.2. The third differential (the first being the series itself) was chosen because a change of sign at this level rather clearly corresponds to a visual event or "boundary" in the series.

There are many spikes of different sizes corresponding to different degrees of sharpness of the turns. It was found from observation that in this web log analysis case it was sensible to take the first turn or 'spike', whatever its frequency. This approach entirely removed any need for parameterization in the boundary discovery but in this case the algorithm was not tested for resolution discovery. In the chapter on TURN*, this becomes a major focus of enquiry.

Varying the number of times the series was differenced was investigated and the third level proved optimal. It is not being asserted that this can or has been proved. However, it is interesting that single and double differencing is a standard method for rendering time-series stationary [43]. This is based on the assumption that the series obeys 'homogenous' nonstationarity. That is to say its behaviour is uniform over the series apart from occasional changes in level or slope. Even though this is a fairly strong criteria, many real world series do in fact exhibit this kind of behaviour [43]. Differencing is a potent

high-pass filter and (sic) helps reveal underlying trends or significant changes.

The interest here is in finding important turning points. In any real world series, there will be many turning points (maxima and minima) and the challenge is to find the most significant ones. A typical heuristic used is to pick turns which are followed by a certain minimum length of trend, i.e. no other turn within $k$ data points. However, study of this during this research found this heuristic unreliable due to being too global and not capturing sufficiently the local dynamics.

The intuition behind the approach of TURN is that turns in the differenced series will indicate a more fundamental change. For example, the change of speed of a car might and typically will fluctuate while accelerating but the fact that it is accelerating is often more important than these minor fluctuations. If the rate of acceleration is studied rather than the speed of the car or the acceleration itself, we can detect the point when the driver moved her foot from the accelerator to the brake.

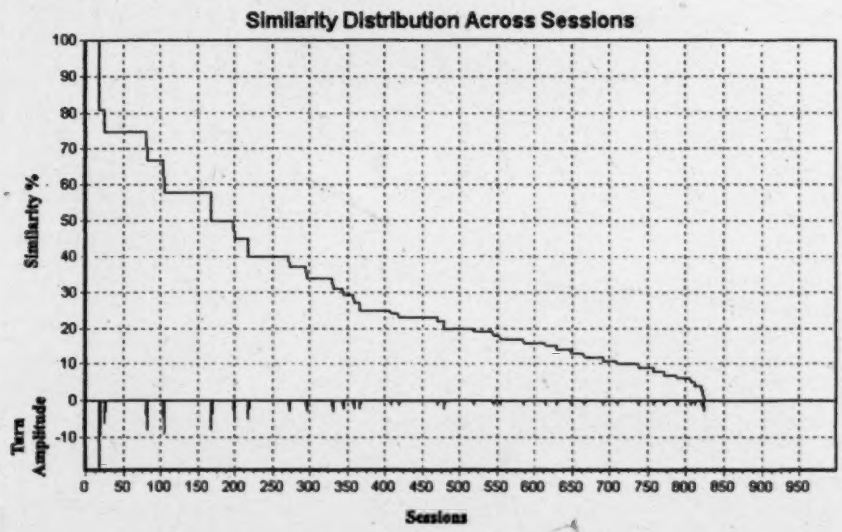



Figure 3.2: Distribution obtained with Jaccard coefficients and turns found by TURN. From [15]. (negative spikes).

Figure 3.2 illustrates TURN applied to a non-Euclidean distribution. The figure shows a typical plot of similarity between one session and 999 others, given 1000 sessions. When TURN finds a turn, it produces a spike with an

amplitude related to the importance of the turn. Here these spikes are shown as negative values. Typically, there are some sessions that have zero distance from (100% similarity to) the reference session and then there is a sharp drop off followed by a series of smaller adjustments until it drops to zero similarity.

### 3.1.3 Comparative Analysis

One recent algorithm which is well suited to non-Euclidean spaces is ROCK[18]. ROCK clusters a sample of the data using an arbitrary similarity measure that returns a value between 0 and 1 and proceeds hierarchically by merging clusters with common neighbours exceeding a given threshold $\theta$ until reaching a fixed number $k$ of desired clusters. It was chosen to compare our algorithm TURN with ROCK since ROCK is especially suited to handling categorical data. ROCK samples the data set because of its high complexity while TURN does not require to sample the data but considers all data points.

## 3.2 TURN Clustering Algorithm

To decide which cluster to assign a data point to, we need to locate the natural boundary between the clusters. If shown a picture with patches of black on it, we would naturally define them as clusters by identifying white or grey areas between the darker ones. This amounts to searching for the minima in the distribution of 'blackness' - turning points. TURN is an attempt to do this and in the case of web log analysis, it seeks to find the sessions/users close to each other by grouping sessions that fall within the range of the first turning point.

The distribution is differentiated $d$ times looking for a change of sign which we take as a 'turn'. In this application where a similarity measure is used and the similarity values are sorted to find the closest items, changes of sign only occur for $d \geq 3$. The series itself is a set of differences ($d = 1$). As they are sorted, the size will increase monotonically so the next differencing will also produce only positive values ($d = 2$). The sign changes for $d = 3$ identify boundary type events as can be seen in Figure 3.2 where the Turn Amplitude

spikes seen in the lower part of the figure correspond to drops in the Similarity seen in the upper part.

Only the first turn was used as beyond that cluster quality declined sharply. This is a result of the algorithm which joins neighbours of neighbours and as the distance between an item and its neighbours increases the cluster spreads out and it becomes increasingly difficult to see why the items have been grouped. Of course, in a "spherical" cluster all points are obviously related. If the "shape" is arbitrary, this is not necessarily so. The TURN algorithm allows for the discovery of arbitrary "shapes" even though there is no metric sense to this, but the simple validation method used here to evaluate cluster quality is biased toward the "spherical" (as is the case for virtually all proposed cluster validation methods as discussed in Chapter 2).

More generally in other applications, all turns found would be ranked to discover meaningful levels of granularity within the data, which may ultimately involve a heuristic to terminate the search. In this present application, beside the choices just listed, the approach is entirely free of parameters. User control is provided to the user by the choice of filters and similarity coefficients.

The version of the algorithm used here is:

## TURN Algorithm

*Input: Categorical data points*
*Output: Clustered data points*

1. Select an unclassified item A as a seed;
2. Compute all distances between A and all other non-classified items;
3. Sort this result;
4. Search through the sorted result until the first "turn";
5. Classify all items up to the first turn into the same cluster as A;
6. Take each item classified as the seed A and iterate recursively 2-6 until no new items are added. Then go to 1 unless all items have been classified;
7. Allocate all clusters with less than 4 members as noise.

The sorting of the difference data introduces an $O(Nlog(N))$ cost. This could be reduced substantially by introducing a heuristic that the first turn must be within 50% similarity and thus any data with similarity $< 50\%$ can be discarded. In resource bounded reality, heuristics are always attractive. Excluding that, TURN's complexity is $O(zNlog(N))$ where $1 \leq z < N$ and $z$ is a function of the number of clusters and their sizes. It goes to $O(Nlog(N))$ when only one cluster is found and $O(\sum_{i=0}^{N-1}(N-i)log(N-i))$ when every point is deemed an outlier.

TURN's memory requirements are $O(N)$. Since applying certain sensible filters can reduce the number of transactions by 75% or more, very large logs can still be held in main memory. Even if the web log is too large to be held in memory, part can be held on disk. As TURN classifies sessions/users these no longer need to be held in memory so at some point in processing, no further disk accesses are required.

Initially TURN was used for two purposes: A) As a clustering algorithm itself and B) as a means of automating parameter determination for ROCK

(i.e., $k$ and $\theta$). This (B) was the original goal but as the work proceeded it was clear that TURN could be developed into a full clustering method. It was also felt that it would be interesting to compare the two as well as using ROCK with some input parameters. We choose two reasonable sets of parameters to test with ROCK before running the tests and to avoid any accusations of bias. No adjustment was made after getting the results.

ROCK [18] is a robust agglomerative hierarchical clustering algorithm, which is particularly suitable for clustering categorical attributes. Since session information is also a kind of categorical attribute, ROCK is suited to clustering web log data. ROCK's cluster similarity is based on the number of shared neighbours and it follows an iterative process of merging cluster pairs on the basis of a measure of best merging 'goodness'. This iteration will terminate when the number of remaining clusters has reached the specified number, or when no further clustering can take place.

ROCK, like TURN can find clusters of arbitrary shape and can be applied in a non-Euclidean space, unlike many clustering algorithms. Its computation complexity is $O(N^3)$ and the memory space complexity of the algorithm is $O(N^2)$. Because of this ROCK clusters on a sample. This and the choice of a threshold for choosing neighbours make ROCK rather unstable. Changes in the parameters can substantially affect the clustering result. It is difficult for the user to predict what parameters are appropriate especially for a non-Euclidean space which is much more difficult to conceive.

For this reason we were looking for a method for finding parameters automatically and initially developed TURN to find parameters for ROCK. While the method we employed for doing this is rather simple, the results as shown in Table 3.4 indicate how TURN has adjusted to the changing of filters and produced quite consistent results. Even the best parameters for ROCK give less consistent results across filter choices.

The method of parameter discovery was to let TURN count the number of turns in the similarity distribution for a sample of the data set and compute the mean of these. As many of these turns may not be important, this gives us a larger figure than ROCK is likely to find but as ROCK stops before

reaching the user defined cluster number when it can not find an improvement by further clustering, this was an effective if not very rigorous approach. We also took the mean of the thresholds found for the first turn and used that as the threshold for ROCK. Optimizing this choice might well improve the results. We compared these results to those for TURN and ROCK given different threshold values.

We developed an interface to give the user immediate access to the functionality with views on the site and on the consequences or results of any action. The interface lets the user open a cleaned web log and look at any session or user and compare it with another and see the similarity computed between them with any coefficient of similarity selected. It is also possible to see the similarity or distance between any one session or user of the web site and all others and the points marked as "turns" by TURN. This can be viewed either in a text window (Figure 3.3) or graphically (Figure 3.2). In Figure 3.3 in the left text area, the interface is displaying the results of the analysis with reference to a single data point (id 0). In the leftmost column, the point ids are given, in the next the dissimilarity from the reference point, and the following two columns are the difference values from the previous column. To the right of this, the cut point taken by the algorithm is shown. Points above the first cut are clustered in with the reference point.

The ability to easily study sessions from the site allows the user to quickly determine sensible settings for the filters, level of generalisation (Attribute Oriented Induction) and other options.

## 3.3 TURN: Results

We used a web log from an on-line university containing transactions of registered learners accessing a variety of courses and on-line activities. After cleaning of the log data, the user of the interface can choose between various similarity coefficients and various filters before clustering. There are a very large number of possible combinations. Of the similarity coefficients, we found the Jaccard coefficient to be the most useful. The only coefficient that pro-

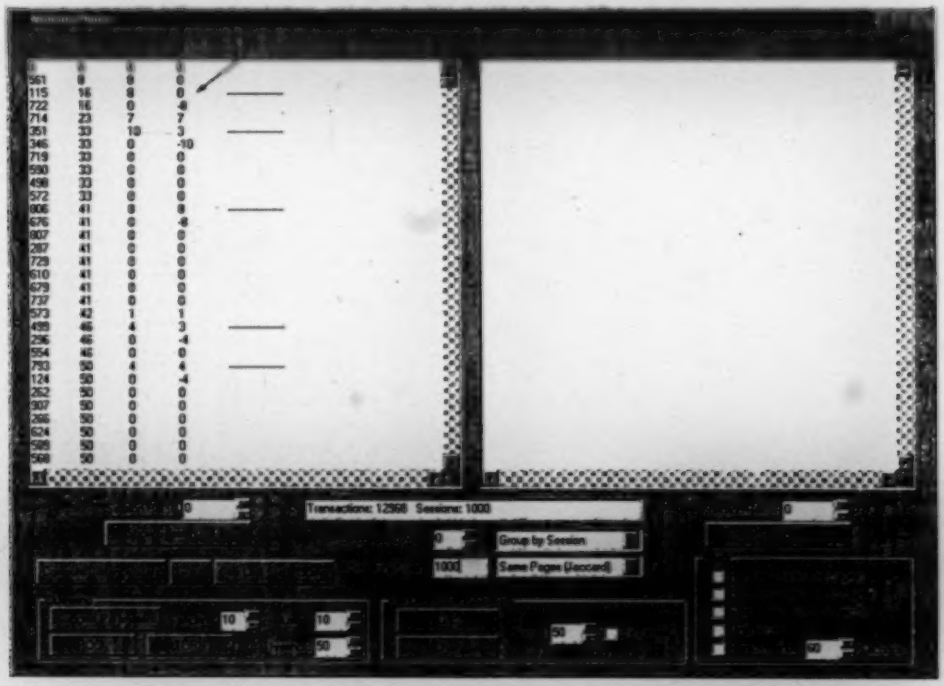



Figure 3.3: Visualization of turning points on the third differential. From [15].

duced significantly different results was the Overlap coefficient and, as we have explained above, this causes very different sessions (intuitively) to be classified as similar. Thus the results presented here are for the Jaccard coefficient which simply computes the intersection of two sets of URLs over their union. Which parts of the URLs and which URLs we select from within any session are determined by the filters.

Three filter combinations were studied:

1 Remove short duration pages ( $< 60$ seconds);

2 Filter 1 plus 'remove duplicate pages'; and

3 Filter 2 plus 'remove duplicates after generalizing to the third level'.

Generalizing to any level is the process of Attribute Oriented Induction discussed above in this chapter ([16]). It lets us group sessions by the visits to a certain nodal level of the site tree, for instance to the course level or

chapter level on an academic site. The filter choices reduced the number of transactions on the test web log by 75.6%, 86.4%, and 92.2% respectively.

In order to assess the clustering of the two algorithms TURN and ROCK, we computed a cluster quality measure. This is simply the mean similarity $\mu$ between all data items within a cluster. This is of a similar form to, though employed differently than, Hubert's $\Gamma$ statistic used for cluster validation (see Chapter 2). That is,

$$\mu = \frac{2}{n(n-1)} \sum_{i,j,i<j}^{n} sim_{ij}$$

where $sim_{ij}$ is the similarity between sessions $i$ and $j$ computed as a percentage using the selected coefficient, and $n$ is the number of sessions in the cluster. Outliers are excluded (cluster size $\leq 3$) to get a meaningful metric of cluster quality. For example, if the metric differences between them are all zero, this gives a mean similarity of 100%.

The issue of validating cluster quality is a difficult issue that is discussed elsewhere in this thesis (Chapter 2, Cluster Validation). The approach adopted in this circumstance would not be appropriate for metric data as it would involve the assumption of "spherical" clusters. In an arbitrary shaped cluster, like a very long thin stick-like one, the similarity between points at either end may be low. However, for non-metric data where a "shape" for a cluster can not be reasonably inferred, it seems most reasonable to fall back on the test of mutual similarity amongst the cluster members. Suffice it to say that even though it is possible to criticize this choice, no one has proposed a demonstrably better solution. Indeed, all cluster validation methods proposed so far suffer from the same assumption [19].

For TURN, we also measured the value of $z$ where the complexity is $O(zn)$ plus the cost of sorting. For the dataset tested, it was found that $n/2 > z > n/3$.

ROCK has no defined means of dealing with outliers. So here it was considered that all clusters with less than three members are outliers for both algorithms. The tables are computed on 1000 sessions with a 10% sampling

rate for ROCK. Both algorithms scale up though ROCK's sampling rate has to be reduced as the data size increases.

| Filters | Cluster Quality (%) | No. of Clusters | No. of Outliers |
|---|---|---|---|
| None | 86.29 | 44 | 574 |
| 1 | 83.64 | 43 | 582 |
| 2 | 93.12 | 46 | 550 |
| 3 | 99.39 | 36 | 122 |

Table 3.1: Clustering Results for TURN.

| Filters | Cluster Quality (%) | No. of Clusters | No. of Outliers |
|---|---|---|---|
| None | 48.50 | 8 | 489 |
| 1 | 61.09 | 26 | 599 |
| 2 | 70.39 | 32 | 127 |
| 3 | 79.98 | 17 | 127 |

Table 3.2: Clustering Results for ROCK using a 40% Threshold.

| Filters | Cluster Quality (%) | No. of Clusters | No. of Outliers |
|---|---|---|---|
| None | 64.87 | 12 | 855 |
| 1 | 90.38 | 18 | 768 |
| 2 | 95.79 | 46 | 320 |
| 3 | 100 | 28 | 180 |

Table 3.3: Clustering Results for ROCK using a 70% Threshold.

Since ROCK requires parameters, two reasonable parameter sets were selected representing a looser and tighter cluster definition. As can be seen in Table 3.3, ROCK's cluster quality was higher with the tighter cluster definition but the results were much poorer when no filters were applied. We also used TURN to suggest parameters to ROCK (Table 3.4) and while ROCK can be fine tuned to get better results with the right filters and parameters, TURN's parameter choices were more consistent across filters.

Further, it is clear that the most consistent results come from the use of TURN as a clustering algorithm in its own right. TURN found more clusters

and left fewer outliers than ROCK, which could be expected from ROCK's clustering being based only on a sample. Small clusters could easily be missed by a sampling process. TURN also had a somewhat higher cluster quality overall and particularly so when no filters were applied.

| Filters | Cluster Quality (%) | No. of Clusters | No. of Outliers | Threshold |
|---|---|---|---|---|
| None | 82.83 | 31 | 638 | 76 |
| 1 | 87.19 | 22 | 699 | 52 |
| 2 | 64.62 | 33 | 519 | 45 |
| 3 | 80.72 | 18 | 73 | 48 |

Table 3.4: Clustering Results for ROCK using TURNs Threshold Parameters.

While ROCK runs faster than TURN due to only clustering a sample, TURN avoids the potential errors due to sampling and has substantially lower complexity and memory requirements.

# Chapter 4

# TURN*

## 4.1 TURN* Algorithm

In this section a new clustering algorithm *TURN** is introduced developed as part of this thesis work. It is a non-parametric clustering approach for metric data that efficiently discovers clusters of arbitrary shapes. Figure 4.1 presents an overview of the different phases of the algorithm.

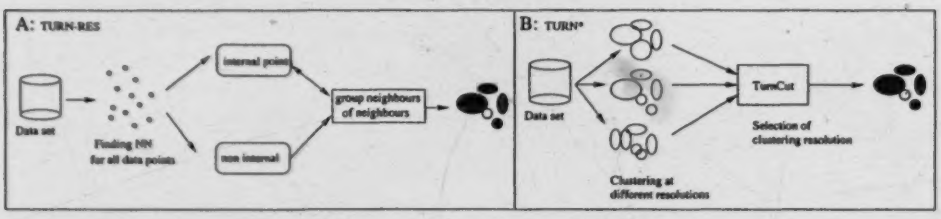



Figure 4.1: A: Steps of the TURN-RES Clustering algorithm. B: The general design of *TURN** using TURN-RES to cluster at different resolutions. From [62].

Any dataset is approximately analog. Crisp clustering, or any kind of compressed representation that discards some of the approximately analog information, must include certain parameters. Thus in this context, non-parametric clustering means that the approach is reasonably robust across varied datasets so that the user is not required to input or adjust any parameters. Later in this section, having introduced the parameters and decisions involved, they are discussed in this light.

**Definitions**

At the end of this section, a general form of the TURN* algorithm is presented and formal definitions given of the concepts involved. This thesis presents an instance of this general form, the application to 2D spatial data. First, the concepts involved are explained with respect to this instance and then the algorithms are given.

The following concepts are used: Resolution, Neighbour, Close neighbour, Local density, Internal point and Optimum resolution. These are defined in a more descriptive way at this stage.

**Resolution:** The resolution or scale of the $x$ and $y$ values of a point depends on a factor by which the original values are multiplied. After the values are transformed, they are rounded down.

**Neighbour:** For a point with a given $x$ value within the grid, the nearest neighbours are the two points with the least difference in the y value. There will be exactly two chosen one with an equal or higher value of $y$ (Right neighbour) and one with an equal or lower value of $y$ (Left neighbour). In the case where more than two qualify, any two can be chosen without effecting the algorithms.

**Close neighbour:** Points p, q with equal $x$ values and and $y_p - y_q \leq 1$ given the resolution, or grid, are considered "close".

To illustrate what is meant by scaling, points $x_0$ (1, 1) and $x_1$ (1, 3) are not close neighbours unless the scale is reduced (e.g. $\div$ 2 giving (after rounding) $x_0$ (0, 0) and $x_1$ (0, 1).

**Local density:** A density value $t_P$ is computed for each point $P$ based on its distance to its nearest neighbours, irrespective of their closeness, which, in this approach, is a maximum of 2 per dimension, one on each side. Where $d_L$ and $d_R$ are the distances to the left and right neighbours along a dimensional axis $i$:

$$t_P = 1 \div \sum_{i=1}^{2} \sqrt{(d_{L_i}^2 + d_{R_i}^2)} \tag{4.1}$$

The distances $d$ are constrained to be a minimum of 1. This forces $t_P$ to

have a finite maximum value which facilitates the next definition.

**Internal point:** $t_P$ allows us to determine how closely packed the points are locally and a threshold $\phi$ is set as a cut-off to differentiate between points that are to be treated as internal or external to a cluster (See Figure 4.6). Thus a point is internal where $t_P \geq \phi$ where $\phi$ is close to its maximum, in short, a little slack is given. It can be seen that this parameter scales with the resolution and is not something to be adjusted by the user. We found the value used to be satisfactorily robust across many differing datasets.

**Optimum resolution:** This is the key breakthrough in this thesis and is also the most difficult to define. An optimum resolution is a resolution that a human observer would recognise as a useful clustering result. It is asserted that this must be associated with an area of stability in the clustering result over some range of resolutions. This is identified by a change in the underlying trend across resolutions signified by a change of sign in the third differential of the statistic studied.

For example, if one is studying something under a microscope and the view is continuously and uniformly changing as one adjusts the magnification, no particular magnification will be fixed upon. However, if at some range of magnification certain structures don't change as the trend of change reverses, then these will be naturally identified by the user as of interest, i.e. a meaningful clustering result. Obviously at different resolution scales, different clustering results may be identified.

### Overview of TURN*

TURN* consists of an overall algorithm and two component algorithms, one, an efficient resolution dependent clustering algorithm TURN-RES which returns both a clustering result and certain global statistics from that result and two, TurnCut, an automatic method for finding the important or optimum resolutions from a set of resolution results from TURN-RES. To date, clustering algorithms have returned clustering results for a given set of parameters, as TURN-RES does, and some have presented graphs or dendrograms of cluster features from which the user may be able to adjust or select the parameters

to optimize the clustering.

TURN* takes clustering into new territory by automating this process removing the need for input parameters. Clustering Validation is a field where attempts have been made to find rules for quantifying the quality of a clustering result. Though developed independently, TurnCut could be seen as an advance in this field which has been integrated into the clustering process.

### 4.1.1 TURN-RES: Clustering at one resolution

This is a fast, efficient, scalable clustering algorithm for a single resolution. While little discussed, except in papers such as [54], resolution is a key concept in clustering. When a radius is defined, as in DBSCAN [10], or some related parameter, a particular view is being set that has an equivalence to viewing a density plot with a microscope or telescope at a certain magnification. The night sky is one example; as the magnification level is adjusted, one will identify different groupings or clusters. The CHAMELEON data set (Figure 4.8) is another example. It looks like there are nine clusters but given a magnifying glass, the large clusters will be seen to have their own sub-clusters.

TURN-RES has one input parameter - resolution. It is resolution or scale dependent because of its definition of close neighbours. Two points are considered "close" only if they are separated by a distance $d \leq 1.0$, at a given resolution, along all dimensional axes. The definitions of neighbour and close neighbour are given above and illustrated in Figures 4.2, 4.3, 4.4 and 4.5. Formal definitions are given at the end of this section.

Points that fall on the edge of a cluster will not be marked as internal but they get included because close neighbours to internal points are pulled into the cluster.

As noise typically can create small clusters, small clusters are flagged as noise or outliers. Small is defined as min(100, $N$/100) where $N$ is the number of data points. It was felt that for small datasets the definition of small had to decrease and for large ones, it should not be allowed to increase as this might cause small but significant clusters to be misclassified as noise. Experiments were made with various values and this choice seemed both sensible in general

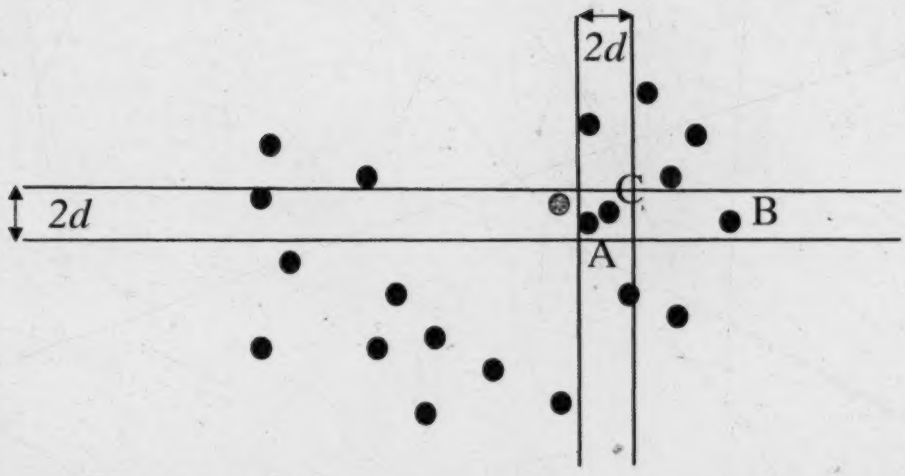



Figure 4.2: A resolution is defined by a distance $d$ along each dimensional axis. At this resolution points A and B are nearest neighbours of point C along the vertical dimensional axis.

and appropriate for the various datasets tested. Since the size and nature of these datasets varied widely, this was taken as indication of the robustness of the choice.

TURN-RES collects certain global statistics or cluster features, being $k$, the number of clusters, $n$, the number of points assigned to clusters (not considering outliers), $t$, total density and $t_m$, mean density ($t \div n$). The interest here is to characterize the resolution level in such a way that levels of particular interest can be determined automatically. $t$ is the sum over $n$ of $t_P$, the local density.

Once the data points have been characterized as internal or external, they are simply agglutinated into clusters. An unclassified internal point is selected, a new cluster number is assigned to it and all of its close neighbours are similarly classified. For each of those that are also internal, the process is repeated. Not incorporating close neighbours of external points stops clusters growing across "noise" bridges.

To quickly determine the nearest neighbour ids of each point a single sort is performed on each dimension. Building the clusters requires one further sort by id. Each sort is followed by a single scan giving a computational complexity

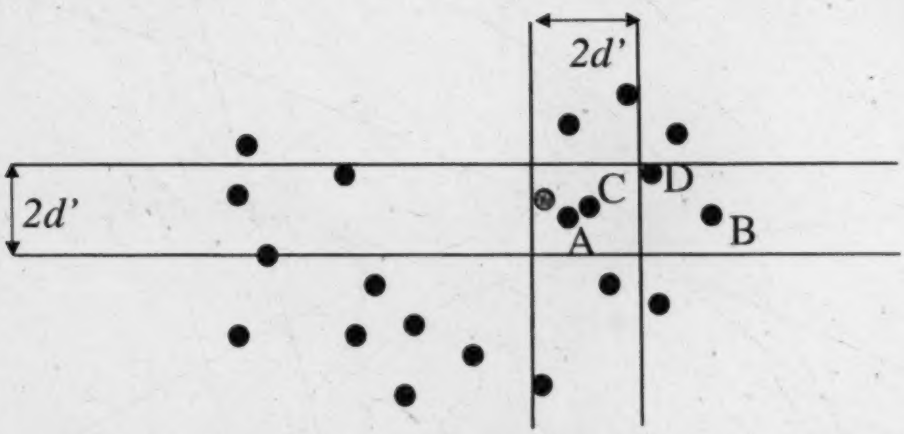



Figure 4.3: At coarser resolution $d'$ the point D now replaces B as the right nearest neighbour of C along the vertical dimensional axis.

of $O(Nlog(N))$.

### 4.1.2 TurnCut

TurnCut uses the core of the TURN algorithm described in the previous chapter, from our previous work in [15], detecting a change in the third differential of a series to identify important areas in a cluster feature across resolution series built by repeated calls to TURN-RES by the *TURN** algorithm.

As already discussed under TURN, single and double differencing are routinely used in time series analysis [43] to render a series stationary. Differencing amounts to a highpass filter which is employed here, differencing thrice (double differencing the change values of the series), as this was found to be the most effective way to reveal meaningful change in the underlying trend.

Though developed independently of work on Clustering Validation, TurnCut automates other authors' concept of finding the "knee" in the cluster feature graph [19]. It picks out the first (and subsequent) "abrupt" change in the overall trend of the curve - acceleration or reversal of the rate of change of the clustering feature studied (see Figure 4.7). If the data points being clustered are homogenously distributed, no "turn" will be found.

If clusters exist, TurnCut will pick out the point where stabilization occurs in the clustering process, which will often coincide with a level that an observer

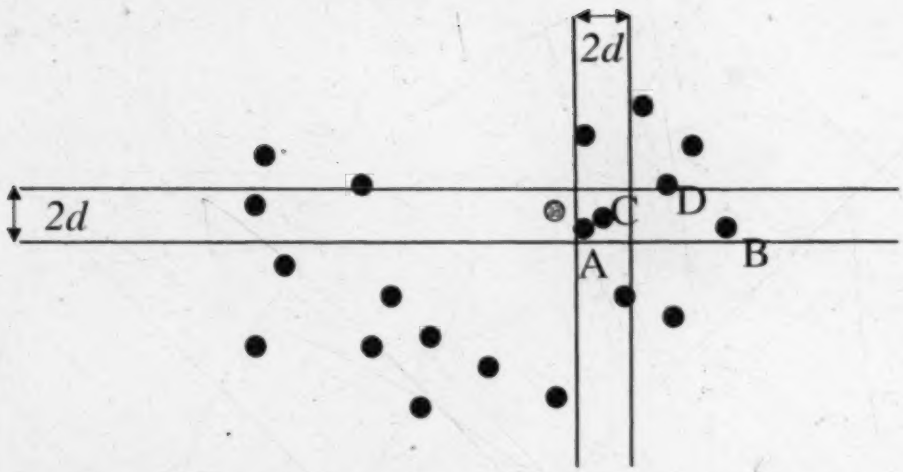



Figure 4.4: At resolution $d$ point A is a close neighbour of C but B is not close: $dist > d$ along the vertical dimensional axis.

would identify as a clustering result (almost by definition - we would never pick out a level that did not appear to represent a certain plateau). In general there can be several of these and the algorithm can find them all even though in this paper the algorithm given for TURN* stops at the first found. In effect, TurnCut is detecting plateaus in the entropy curve.

Other authors only analyzed a feature versus cluster number ($k$) graph [19]. We evaluated applying TurnCut to the $k$ graph and found better results with mean density and often further improvements with total density (as defined above) across resolutions. In most algorithms data can only be collected for a particular value of $k$. In TURN* we can study as fine a resolution step size as we choose, several steps often yielding the same value of $k$.

Our result makes sense because the density statistics have more information than $k$ which is a fairly coarse statistic compared with those we defined that change with every point added to the clustering result. Total density also gives weight to the total number of points clustered which has been factored out of the mean density.

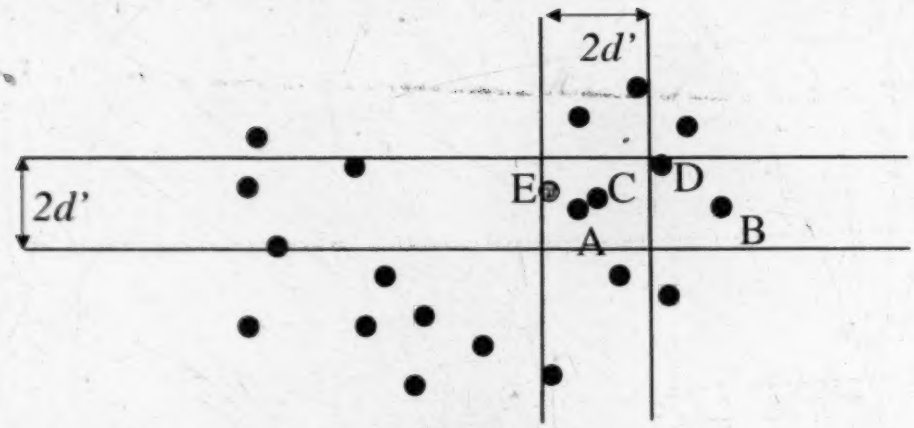



Figure 4.5: At resolution $d'$ both points A and E are close neighbours of C.

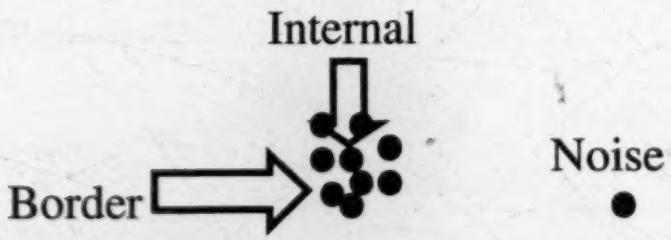



Figure 4.6: Points are differentiated according to their local density and close neighbourhood to internal points.

## TURN-RES Algorithm

*Input: 2D data points and resolution level r*
*Output: Clustered data points*

1. For each data point $P$, scale coordinates of $P$ to resolution $r$ and find the two nearest neighbours on each dimensional axis, and the distance $d$ of each from $P$; if $d \leq 1.0$ assign the point as a "close" neighbour;
2. compute the density $t_P$; set $P$ as internal if $t_P > \phi$ ($\phi$ is fixed and not an input parameter)
3. For each non assigned internal data point $P$ do
   (a) add $P$ to new cluster $C$;
   (b) add all "close" neighbours of $P$ to $C$;
   (c) for all $m$ added points that are internal add all their "close" neighbours to $C$ and repeat until $m = 0$;

### 4.1.3 TURN*: Finding the best clustering

The algorithm proceeds by detecting clusters across a range of different resolutions stopping (or at least flagging) what is considered as the optimal clustering using TurnCut. A resolution is simply a scale by which all data point values are multiplied (see TURN-RES above).

Naturally, TURN-RES will return a clustering result only within a certain range of resolutions. On one end of the range every data point will be classified as noise/outlier. Moving in the direction of increasing numbers of clusters found, the first resolution level at which this occurs is called $S_\infty$ here. Moving in the other direction a situation is reached where all points are included in a single cluster ($S_1$).

First, the algorithm seeks $S_\infty$ starting by clustering with TURN-RES at resolution 1:1 and then stepping out at a large geometric increment ($p_{2.5} = \times 2.5$ where the scale $>$ 1:1; $\div 2.5$ where the scale $<$ 1:1) clustering until $S_\infty$ is found. Then the step size is reduced ($\times .5, \div .5$) and a scan over $S_\infty \rightarrow S_1$ is performed. Geometric steps sizes are used as: a) this ensures quickly finding $S_\infty$ and traversing the range to $S_1$, and b) optical magnification steps are always given in geometric values.

At each step, TURN-RES is called and the clustering result and global features/statistics are stored and TurnCut is called to assess if an optimum clustering has been achieved. If so, either TURN* stops or it can be allowed to continue, collecting information for all the key resolutions flagged by TurnCut. This is illustrated in Figure 4.7 where the various statistics collected have been normalised to stay in a range equivalent to the variations in $k$, the number of clusters found. This is valid since the interest is in the shape of the curve, not the absolute values.

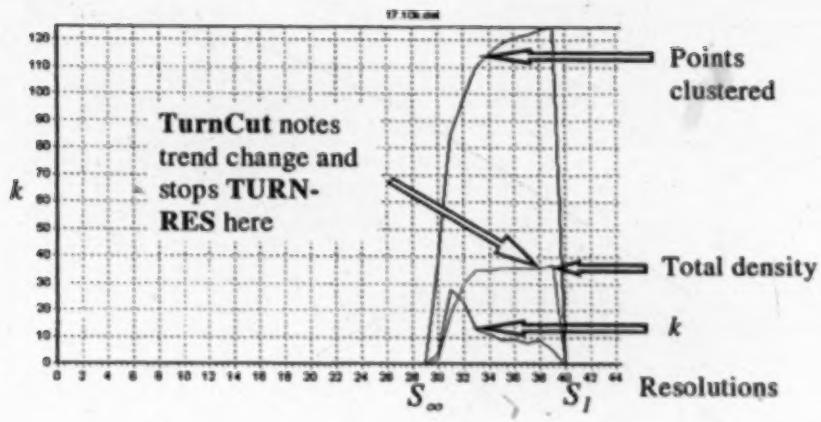



Figure 4.7: TurnCut finds the optimum clustering resolution for CHAMELEON data set t7.10k.dat.

## *TURN** Algorithm

*Input: 2D data points*
*Output: "Best" clustering of data points*

1. start at resolution $r$=1:1. Seek $S_\infty$ by increasing/decreasing the resolution by step $p_{2.5}$ (multiplying or dividing by $2.5, 5.0, ...$) and clustering at each resolution until all points are labelled as outliers;
2. scan from $S_\infty$ towards $S_1$ ($k = 1$) by repeating
   (a) decrease the resolution $r$ by step $p_{0.5}$;
   (b) cluster at $r$ with TURN-RES;
   (c) store clustering result and statistics $t$, $t_m$, $k$, and $n$ for $r$;
   (d) stop if $k = 1$ else call TurnCut to determine if $r$ is an optimum clustering result and stop on success

No sampling takes place, however the scalability of TURN* is evidenced by the performance on small and large data sets (Table 4.2 and Figure 4.17).

As already shown, the time complexity is $O(Nlog(N))$. The space complexity is straightforward. For each object, a simple data structure is needed to store coordinates in the $D = 2$ dimensional space, the nearest neighbours on each dimensional axis, and some specific data such as cluster label, type of point etc. Thus, the memory space needed is $O(DN)$.

### 4.1.4 Parameter Free?

The parameters involved in the component algorithms are 1) $\phi$, that defines if a point is to be treated as internal, 2) the resolution level given to TURN-RES, 3) the definition of a "small" cluster, and 4) the step size(s) used between resolutions at which TURN-RES is run.

This implementation is parameter free in so far as 1) $\phi$ is part of the concept of closeness for a resolution level and thus should not need to be varied as proved to be the case; 2) TURN* feeds TURN-RES a series of resolutions starting from the extreme case ($S_\infty$) where all points are identified as outliers so the user is never asked to enter a resolution; 3) it was found that only very small data sets would need this modifying and TURN* is not intended for such sets; 4) the step size starts large until $S_\infty$ is found and is then reduced.

In choosing the step size there is a trade-off between fineness and speed. We found the step size we chose to be robust across varied data set types but if the differentiation between cluster densities in the data was small, a key resolution could be missed. However, to be secure against this the algorithm could be extended as follows: Once TurnCut flags a resolution of interest the range across the previous step can be scanned by further reducing the step size giving finer resolving power. It is most unlikely that any user input would be needed in this case and the speed of the algorithm makes this additional processing reasonable.

While there will never be a perfect parameter free solution, this implementation proved robust across a wide range of different data set types with differing cluster densities, closeness of clusters, arbitrary shaped clusters and noise levels including many examples of "noise bridges", places where noise formed apparent bridges between clusters as is obvious in the figures showing

the results on the CHAMELEON dataset.

### 4.1.5 Generalising and Formalising TURN*

In this section is presented a generalisation of TURN* with formal definitions and some discussion of related work.

#### Definitions

Given a dataset $S$, a set of orthogonal attributes (dimensions) $A = \{a_0, a_1, ...a_k\}$ where each $a_i = \{x_0, x_1, ...x_k\}$, a set of resolutions $R = \{r_0, r_1, ...r_k\}$ across $A$ ($|A| = |R|$) and points $p, q \in S$. Note that the use of the minus symbol in some of the definitions represents some function which yields an equivalent result appropriate to the domain. For example, in the case that each $x_i$ was in fact itself a set or where a correlation exists between attributes as in the case of a dataset of shapes existing in several dimensions. The result is a distance function $D(p, q)$ equivalent to $x_p - x_q$ in the straightforward ordinal case. All such comparative symbols can be appropriately overloaded.

**Resolution:** a resolution is defined by its unit distance $u$ where $\{x_{i+1} - x_i | a_i, r_i\} = 1.0$, where a value $x'$ in the original untransformed dataset is related to the transformed value $x$ under resolution $r$ such that $x = x' f(u)$ and all $x$ are rounded down.

**Left Neighbour:** if $S$ is sorted cyclically on $A$ in the order $a_i, a_{i+1}, ...a_{i-1}$, the left neighbour of $p$ is $p_L \in S$ where $\forall q \in S$ with $x_q \leq x_p$ $\{x_p - x_{p_L} \leq x_p - x_q | \{\forall a_j, j \neq i, x_p - x_{p_{LC}} < 1.0 | r_j\}, a_i, r_i\}$.

**Left Close Neighbour:** $p_{LC}$ obeys $\{x_p - x_{p_{LC}} \leq 1.0 | \{\forall a_j, j \neq i, x_p - x_{p_{LC}} < 1.0 | r_i\}, a_i, r_i\}$. There may be many left close neighbours and any one and only one is chosen. Which is chosen has no effect on clustering because all will be close neighbours of each other and thus clustered together.

**Right Neighbour and Right Close Neighbour:** $p_R$ and $p_{RC}$ follow in the same manner as the above.

**Local Density:** this is given by the function $Tfunct = \sum_A f(D(p, p_L), D(p, p_R))$. That is, a sum is made across all attributes (unless clustering is being restricted

to a subspace) of a function of the distances from $p$ to its left and right neighbours.

**Internal Point:** $p_{int}$ where $\{Tfunct_p \geq \phi | A, R\}$ where $\phi$ is a threshold value close to its maximum.

**External Point:** any point that is not internal is external.

**Connected Points:** Two points $p, q \in S$ are in the same cluster $C$ iff there exists a chain of points $p, p_1, p_2, ...p_i, q$ such that every point is a close neighbour of its neighbours in the chain and no interconnecting points are external. At least one point in the chain must be internal. This definition is both transitive and symmetric. Thus it follows that the same clustering result will be found whatever point one starts from in assembling the clusters so TURN-RES is independent of the order of the points presented That is, it follows that we can obtain a cluster $C$ by randomly selecting an object $p$ $\forall p \in C$. It also follows that the set $C$ is maximal.

**Clustering:** Thus a clustering $CL$ of a dataset $D$ consists of a set of clusters $CL = \{C_0, C_1, ...C_k\}$ where each $C_i$ is a set of connected points.

**Noise:** Noise are those members of $CL$ such that $|CL| < MinSize$ however defined plus all those points with no close neighbours.

### TURN* and GDBSCAN

An interesting evolution from DBSCAN is the generalised form GDBSCAN [53]. The authors generalise DBSCAN's two parameters $\epsilon$ and $MinPts$ by replacing the distance radius $\epsilon$ by any arbitrary binary reflexive predicate $NPred$ and $MinPts$, the density threshold, by an arbitrary weighted cardinality $WCard$ of the local set of objects satisfying $NPred$. An example of $NPred$ would be "intersects" in the case of polygons, showing how GDBSCAN can cluster nodes with features, such as any shape.

Our approach in TURN* as presented here requires each node to be defined by a continuous set of values per attribute on which clustering is to take place, with a clearly defined sequence and hence conceptualization of neighbourhood. For example, in the set of upper-case letters, B follows A. Sets exist which do not meet this criterion, viz { apple, orange, pear }. In such a case, one can

group all the apples together separate from the oranges but to construct a meaningful cluster containing apples and oranges but not pears would require some concept of proximity of oranges to apples versus pears to apples. Thus we see informally, by example, a general result that to form meaningful clusters of objects holding more than one instance of the set of attribute values per attribute requires the neighbourhood concept between values required by TURN*. More formally, given an attribute $a$ consisting of the set of values $X$, if $\phi_a$ operates on an object $p$ to generate a value $x \in X$ and $\theta$ is an associative operator on a set of objects $D$ w.r.t. $a$, then, for any $p, q \in D$, $\{\theta(p,q)|a\}$ returns a value iff $\theta(\phi_a(p), \phi_a(q))$ returns a value. If this were not true, a relationship between two objects w.r.t. $a$ could be determined without knowledge of the relationship between values in $X$, which is absurd.

A more challenging case is where single values per attribute are replaced by sets of associated values across attributes, as in the case of shapes - e.g. polygons. The simple sort approach employed here, which amounts to the projection of a global grid on the data in order to establish the nearest neighbours of a point would have to be modified to order the attribute sets in a meaningful way. The quicksort algorithm already takes an arbitrary evaluation function, say $NPred(a)$ where $a$ is the dimension or attribute along which the ordering is to be returned. $NPred(a)$ is a binary reflexive predicate such that for dataset $S$ and all $p, q \in S : \{Order(p,q)|a\} = NPred(a,p,q)$. Then the determination of distance would be computed using any appropriate function $\{D(p,q)|a\}$. As defined above, neighbours and close neighbours are determined, a local density $t_p = Tfunct$ established and a reflexive predicate returns whether a point is internal or external based on the local density. An internal point is equivalent to GDBSCAN's "core" point. The entire computation is for a given resolution set $R$ as defined above.

$R$ is related to GDBSCAN's $NPred$ and $Tfunct$ is related to $WCard$ but differ as TURN* operates explicitly on each attribute $a_i$ before combining the attribute-dependent results. It should be noted that all attribute values needed to determine attribute-dependent results are passed but only the value relating to $a_i$ is returned. This allows TURN* to scale independently across

attributes. In TURN*, the definition of connected points is both transitive and symmetric. In the equivalent case of density-connected in GDBSCAN, the definition is transitive but not necessarily symmetric however the authors show, with further analysis, the same results of order independence and maximality hold.

GDBSCAN and the above definition of TURN* are highly general but TURN* differs fundamentally as GDBSCAN, like DBSCAN, inserts threshold parameters into its chosen predicate and thus gives a parameter dependent result. TURN* scans across global (though possibly attribute dependent) resolution values $r$ using TURN-RES and isolates significant values of $r$, giving a parameter independent solution.

## 4.2 Experimental Results

In this section, experimental results are presented to evaluate *TURN** and compare the performance of the algorithm with other well-known clustering algorithms: k-Means, DBSCAN, CURE, ROCK, CHAMELEON and WaveCluster. The implementation of DBSCAN was obtained from the authors of the algorithm (University of Munich, Germany), while the implementation of ROCK and CURE was obtained from the authors of CHAMELEON (University of Minnesota, USA) written for the evaluation of their own algorithm [38]. However, as they could not provide the CHAMELEON code for legal reasons, this was implemented locally. It seems reasonable to believe that the implementation and optimization of CHAMELEON is similar to the one published by the authors in [38] since we get the same performance on the same data sets as that presented in [38]. WaveCluster [55] was also implemented locally for the same reasons with equal success.

TURN* is tested here on data sets provided to us by the developers of CHAMELEON [38] and WaveCluster [55]. Due to CHAMELEON's computational intensity, their files are rather small: 8K to 10K. The main benefit claimed in the WaveCluster paper is its effectiveness on large data sets and their data is 100K-500K+ points. These data sets were chosen because they

are publicly available and they have been used to evaluate other algorithms. Moreover, these data sets are 2-dimensional making it easier to visualize and provide comparisons. We have experimented with various data sets with sizes varying from 8k to more than 575K data points and our algorithm performed well in all cases.

On a 10K data set (t7.10k) Figure 4.8, TURN-RES computed a single resolution in 0.26 seconds and the total process of *TURN** to find the optimum resolution took 3.90 seconds. A single run of CHAMELEON took 28 minutes with the parameter *MinSize* set to 4%. *MinSize* is the size of the graph partition in the first phase of the algorithm. Selecting a different value would slow down CHAMELEON or degrade the results. The process of finding the correct parameters to give a good clustering result took several hours.

DBSCAN is nearly as fast as *TURN** for a single resolution/parameter setting but also required many runs to find the optimal input variables as it is rather sensitive to its parameters. Indeed, in the CHAMELEON paper [38], the authors failed to find a parameter set that allowed DBSCAN to correctly identify all the clusters in their data sets.

For the data sets chosen, *TURN** took typically 10-20 resolution tests, performed automatically, to find an optimum resolution. In our research, *TURN** found the resolution that a human observer would tend to choose in 80% of cases and, in the other cases, it stopped one or two resolutions away. While this sounds like a "miss", albeit close, careful inspection shows that TURN* did find a significant resolution level but the wide variations in cluster densities in the data set meant that no particular resolution could be said to be perfect. In fact, the CHAMELEON authors created these data sets quite intentionally to show off their algorithms ability to handle such a situation vis-à-vis other algorithms.

TURN* can also be allowed to keep clustering beyond the first optimal level found, collecting data at each "turn" or key resolution level building the equivalent of a dendrogram that reveals the clustering structure at different densities which could then be combined or presented in some way to the user though this is beyond the scope of the present work.

As can be seen from Figures 4.10, 4.11, 4.12 and k-means, CURE and ROCK perform relatively poorly on these difficult data sets due to the complex shape of the clusters and the large amount of noise. DBSCAN, CHAMELEON and *TURN** work well. WaveCluster, after much tweaking of its settings, came close to finding the visually obvious clusters. DBSCAN proved very sensitive to the parameter settings. The CHAMELEON authors and others (e.g. the WaveCluster paper) have presented DBSCAN as failing where clusters are connected by noise 'bridges', but we found that parameter settings could be found to deal with this problem in some cases.

However, in actual practice where the user does not know the clustering result in advance, the requirement of finding the right parameters is a major difficulty. CHAMELEON proved more robust on the parameter settings once the right range had been found but it requires the setting of the number of clusters to be sought, which is generally not known. It also fails to separate out noise. Combined with its high complexity, which prevented us from testing it on large data sets, this makes it a weak contender.

*TURN** provides fast, efficient, scalable clustering and identifies outliers allowing for the optional removal of noise as has been done in the *TURN** output presented in Figure 4.9. This would be useful in many applications such as OCR preprocessing, image enhancement, etc. It can stop at or flag interesting levels of granularity identified by the behaviour of global clustering features as discussed.

Here the experimental results are shown of each algorithm on the DS4 data set from the CHAMELEON paper [38], also known as t7.10k. This data set was chosen because it has several features which challenge a clustering algorithm. It has nine clusters of different shapes, sizes and orientations, and the density within and between the clusters varies. In addition, there are clusters within clusters, non-spherical shapes and a large amount of random noise which could create artificial "bridges" between the clusters. In all the clustering result figures, black points indicate data points identified by the algorithm as noise.

### 4.2.1 Clustering Effectiveness Comparison

Clustering results of *TURN** are shown in Figure 4.8. The *TURN** progam identifies small clusters as noise. In addition it correctly identified the nine principal clusters as shown in Figure 4.9 (here all previous users of this dataset are followed (e.g. [38]) in assuming there are nine clusters). This cluster result shows that *TURN** can effectively identify all the 9 clusters and filter out noise. This result was found by *TURN**'s automatic resolution scan process and did not require any parameter tuning.

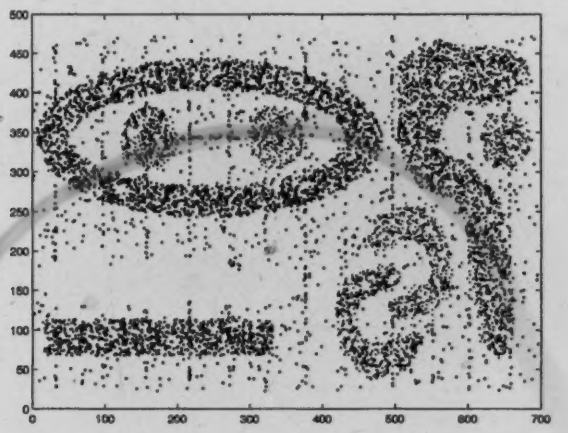



Figure 4.8: *TURN**'s clustering result on t7.10k.dat before cleaning. From [62].

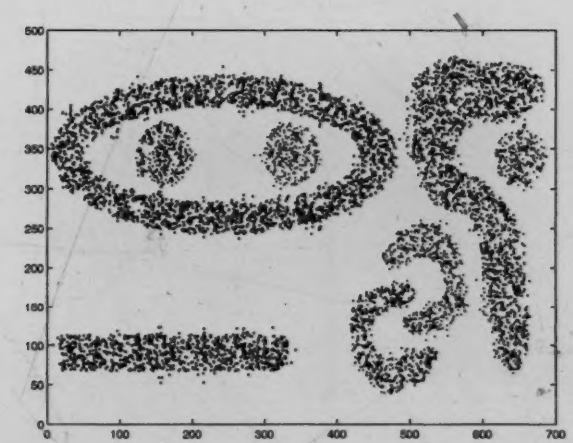



Figure 4.9: *TURN**'s cleaned clustering result on t7.10k.dat after removal of points identified as noise. From [62].

K-mean's result is shown in Figure 4.10. From here we see that k-means tends to find spherical clusters. It is obvious that it is not well suited to find arbitrary shaped clusters.

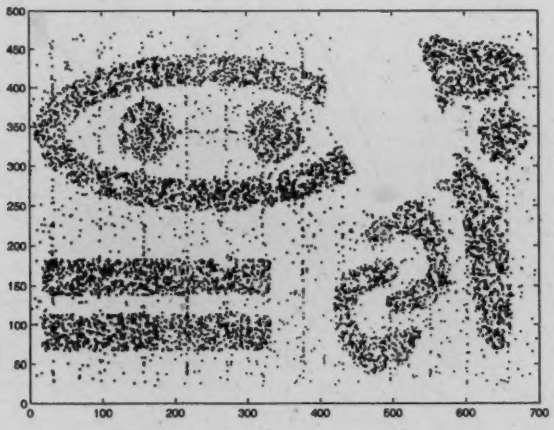



Figure 4.10: K-means's clustering result on t7.10k.dat with $k = 9$. From [62].

CURE is one step from k-means: it uses multiple points to represent a cluster instead of using only one point in k-means. These experiments showed that CURE has similar problems to k-means: it tends to find spherical clusters, although the problem is not as serious as with k-means's. It is difficult for CURE to handle some of the clusters in the test set due to their shape as can be seen in Figure 4.11.

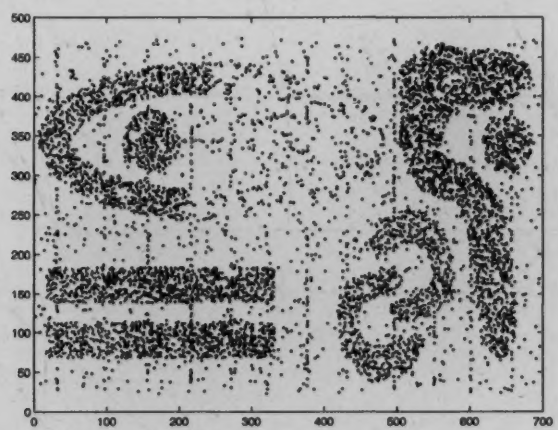



Figure 4.11: CURE's clustering result on t7.10k.dat with $k = 9$, $\alpha = 0.3$, and number_of_representative_points = 10. From [62].

ROCK is designed for clustering categorical data but can, in theory, handle numerical data like the *t7.10k.dat* data set. Its result for clustering this spatial data set is, however, not good. After adjusting the parameters for a long time, the best clustering result found is presented in Figure 4.12. For this result, it was necessary to set the number of clusters to be 1000, and among the resulting 1000 clusters there were five large clusters. The remaining 995 can be considered as noise and they all group around the upper crescent in Figure 4.12. This is because ROCK does not consider noise in its clustering process.

This emphasizes again the problem for setting the number of clusters to be found. Even if the user knows that there are nine clusters, in this case a much larger number needs to be given. If we set the number of clusters to be found to nine, ROCK puts most of the points (9985) in one cluster, and the other 15 points exist in eight noise clusters.

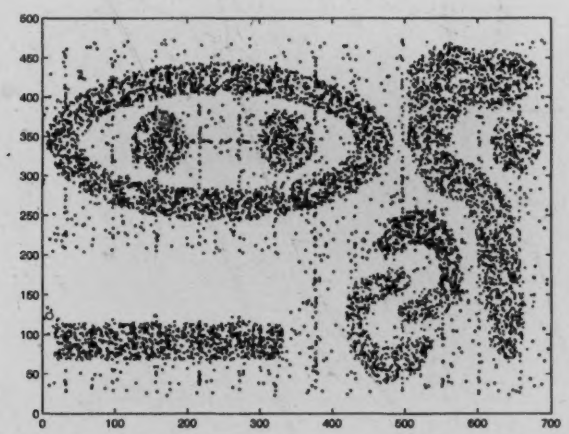



Figure 4.12: ROCK's clustering result on t7.10k.dat with $\theta = 0.975$ and $k = 1000$. From [62].

CHAMELEON's result is shown in Figure 4.13. This result is very close to the result in the CHAMELEON paper. Compared with *TURN**'s result, we see that CHAMELEON does not identify noise. In fact, all the noise points are included in the neighbouring clusters. In addition, we needed to set several parameters for CHAMELEON to obtain this quality of clustering.

DBSCAN gives the good result shown in Figure 4.14. This result is very close to *TURN**'s result in Figure 4.8. We found that the only problem of

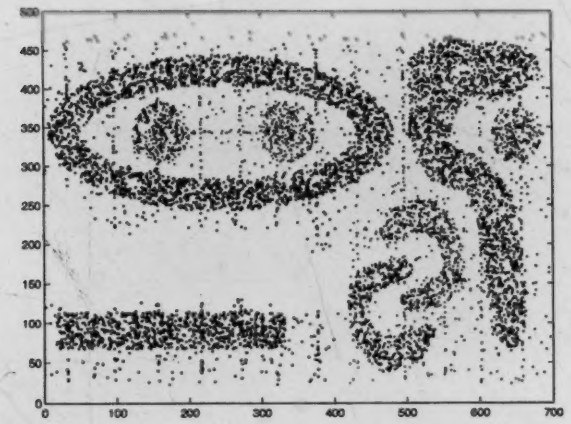


Figure 4.13: CHAMELEON's clustering result on t7.10k.dat with $k = 9$ and *nb_clossest_neighbor* $= 10$, MinSize $= 2.5\%$. From [62].

DBSCAN is that it is very sensitive to the two parameters $\epsilon$ and $MinPts$. For example, if we change $\epsilon$, the neighbourhood radius, from 5.9 to 5.5, it gives a rather poor result (Figure 4.15) due to the density variations in the data point distribution of the clusters. Moreover, if we increase $\epsilon$ and $MinPts$, the noise creates artificial bridges that may cause genuine clusters to merge.

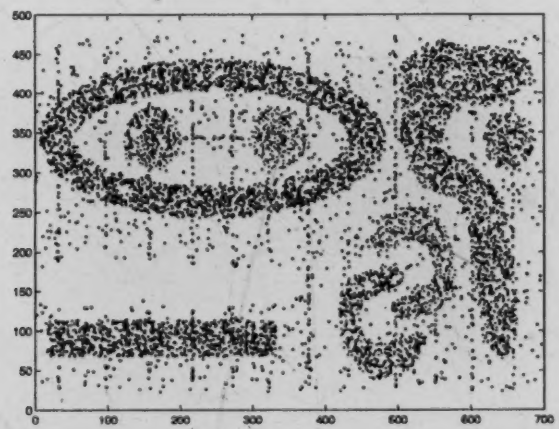


Figure 4.14: DBSCAN's clustering result on t7.10k.dat with $\epsilon = 5.9$ and MinPts $= 4$. From [62].

WaveCluster's best result on t7.10k.dat was with the signal threshold on the transformed frequency domain $\tau = 1.5$ and *resolution* $= 5$ (Figure 4.16). Two clusters are joined due to the strength of the bridge in the averaged

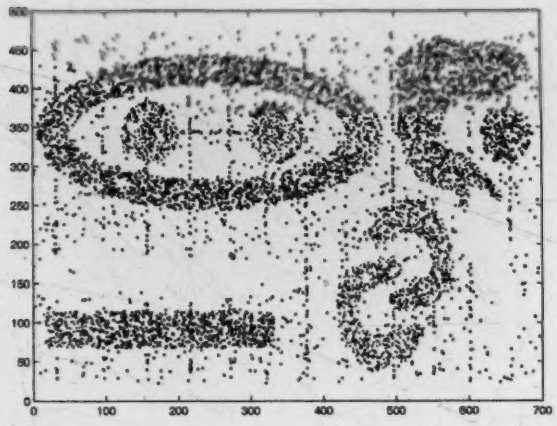


Figure 4.15: DBSCAN's clustering result on t7.10k.dat with $\epsilon = 5.5$ and MinPts = 4. From [62].

signal output. Adjusting $\tau$ (for example) resolves this problem but breaks other genuine clusters.

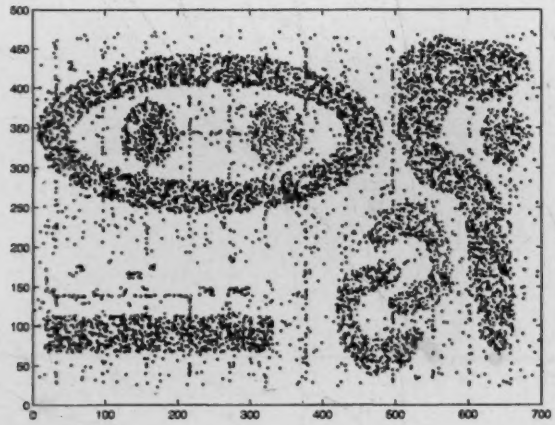


Figure 4.16: WaveCluster's clustering result on t7.10k.dat with *resolution* = 5 and $\tau = 1.5$. From [62].

### 4.2.2 Cluster Efficiency Comparison

The memory complexity for all algorithms is $O(N)$ except for ROCK ($O(N^2)$) and CHAMELEON ($O(kN)$). We recorded the clustering time and memory required for each algorithm. Although this will be related to a certain degree to the implementation coding, the results can reflect characteristics of each al-

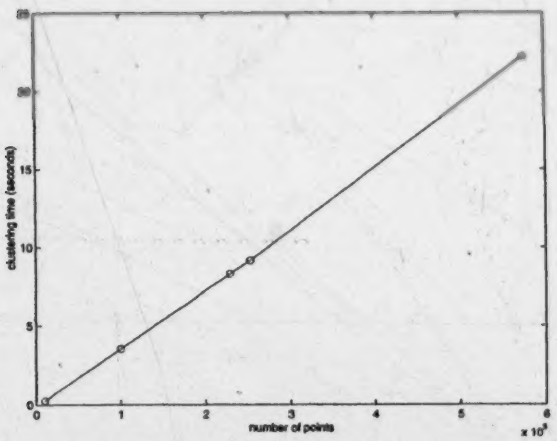



Figure 4.17: Speed curve of *TURN**

gorithm. The implementations of k-means, CURE, DBSCAN, CHAMELEON and *TURN** were tested on a 800MHz Pentium III with 256M memory. Due to its very high memory requirement, the ROCK code was tested on a 1.5GHz Pentium IV with 1G memory. The experimental results are shown in Table 4.1. The following comments on the performance of each algorithm:

- **K-means.** Technically, k-means is the simplest algorithm, and it does not need a complex structure to store data. Consequently, k-means has very good speed and acceptable memory efficiency (less than 9 seconds and 5.5 MB). While the other algorithms were compiled C/C++ code, this implementation was in Java so the speed is not directly comparable. However, it happens to be completely inefficient on this complex data set.

- **CURE.** CURE is the quickest among the three hierarchical clustering algorithms, and it has the smallest memory space (2 minutes 35 and 4.6 MB). This is because it has a relatively simple similarity measurement and simple data structure when compared with ROCK and CHAMELEON. However, legitimate clusters are incorrectly split or merged and more data points are labelled as noise than there is noise in the data set.

- **ROCK.** ROCK is relatively slow (8 minutes and 46 seconds), and its memory requirement is exceptionally large (1.14 Gb). This is due to retaining noise points and treating them as individual clusters taking up memory and CPU time. Each cluster needs to keep an ordered heap of all the other clusters according to the merging goodness, so ROCK's memory requirement is $O(N^2)$. One suggested improvement for ROCK would be to keep only a heap of the $k$ clusters with the highest goodness.

- **CHAMELEON.** As is typical for hierarchical clustering algorithms, CHAMELEON is slow (27 minutes 47 seconds). Because this program is locally developed as the authors' code was not available, some parts of it may not be efficient but still it cannot be expected to be much faster than the other two hierarchical algorithms even after optimization. The memory requirement ($O(kN)$), due to the partitioning of the k-nearest neighbour graph, reached 8.6 MB.

- **DBSCAN.** Among all the algorithms we compared to *TURN**, DBSCAN has the best efficiency. Because of the usage of R*-tree [3], it is efficient in both memory and speed (1.4 MB, Phase I took 5.02 secs and Phase II took 5.51 secs). We wanted to test DBSCAN against *TURN** using larger data sets, but unfortunately the DBSCAN code we got from the authors ran so slowly on the larger data sets (100K or more) that it proved impossible to get any results.

- **WaveCluster.** Due to its grid based approach compounding the data points into a small number of grid cells, WaveCluster is fast completing clustering at the chosen resolution in 0.82 secs. It is also memory efficient consuming 0.8MB

- **TURN*.** *TURN** was the fastest to cluster the 10k data set at 3.9 seconds total, averaging 0.26 seconds for clustering each resolution level. It was faster on a single resolution and, since no human tweaking was necessary to find the right parameters, vastly faster to find the optimum

result. The data structure needed was 140 bytes per data point totaling 1.4 MB.

*TURN** was applied on both the CHAMELEON data sets [38] and the large data sets available from the WaveCluster authors (100K - 575K points) [55], and the test result curve is shown in Table 4.2 and Figure 4.17. From this figure it is clear that the algorithm is both fast and nearly linear with the data set size.

| Algorithm | Clustering time (secs) | Complexity | Memory Usage |
|---|---|---|---|
| K_means | 8.44 | $O(N)$ | 5.5MB |
| CURE | 155.59 | $\geq O(N^2)$ | 4.6MB |
| ROCK | 526.19 | $> O(N^2)$ | 1.145GB |
| CHAMELEON | 1667.86 | $> O(NlogN)$ | 8.6MB |
| DBSCAN | 10.53 | $O(NlogN)$ | 1.4MB |
| WaveCluster | 0.82 | $O(N)$ | 0.8MB |
| *TURN-RES* | 0.26 | $O(NlogN)$ | 1.4MB |
| *TURN** | 3.90 | $O(NlogN)$ | 1.4MB |

Table 4.1: Clustering Speed and Memory Size Results on a data set with 10,000 data points

| Data Set Size | Clustering time (seconds) |
|---|---|
| 10,000 | 0.26 |
| 100,000 | 3.70 |
| 228,828 | 8.29 |
| 275,429 | 9.15 |
| 574,499 | 22.18 |

Table 4.2: Average Clustering Speed of TURN-RES on one resolution with different data set sizes

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion

While many intelligent solutions to clustering data have emerged, it is only recently that researchers have started to address the need for parameter free clustering. Clustering methods that require parameter input will be of little practical use as users will often not know how to provide these nor have the time to acquire them by trial and error use of the clustering method. However, automatic clustering means that the thresholds chosen, as inevitably there are, must be robust across varying types of data sets, varying by size, dimensionality, and density distributions.

This thesis has presented a new fast, efficient and scalable clustering method (set of algorithms) which has outperformed many of the latest state-of-the-art clustering methods already published. In the process of assessing TURN and TURN*, this research has confirmed the weakness in the older methods and the relative benefits of the more recent algorithms. While OPTICS, WaveCluster, and other algorithms provide information at different resolution levels through dendrograms or related graphs, it is left to the user to decide what resolution to choose. Also, all the algorithms have certain choices - parameters - to set. While OPTICS and WaveCluster move some way towards automation they still leave the user with choices to make.

The algorithm TURN handles the challenging case of categorical data in a non-Euclidean space. TURN*, on the other hand, is suited to spatial data or any data for which a metric space can be defined. TURN* can build clustering

information for a data set across resolutions including the equivalent of the dendrograms built by other algorithms. The TURN* algorithm allows us to automatically identify and, if desired, stop on the important resolution level(s) for clustering.

With the proposed extensions discussed in Future Work below, the TURN* family of algorithms represents a complete automated clustering solution. It is fast, scales well with increasing data size, discovers clusters of arbitrary shape and is free of input parameters. It is well suited to a parallel implementation making it even faster with a near linear speedup.

While we have found TURN*'s automatic clustering to be quite robust across a diverse collection of data sets, it is clear that in many cases the optimum clustering resolution is quite subjective and automating clustering in general is highly non-trivial. Clustering Validation is a field that attempts to define standards and methods for this and our algorithm could be seen as a quite successful implementation in this field.

## 5.2 Future Work

TURN* builds information of the clustering at many different resolution levels. From this we can assign fuzzy cluster membership to data points. Fuzzy membership means several cluster assignments with different weights for a given data point. This process (Fuzzy-TURN) will be explored in future work allowing for the discovery and presentation of clusters with widely varying densities including clusters embedded within clusters, which would pose problems to most of the multi-density approaches proposed so far.

An important extension will be testing the TURN* algorithm in higher dimensions. It is designed to scale and it will be interesting to investigate the various challenges posed by high dimensionality as touched on while reviewing other work above. The algorithm is suited to dimensionality reduction if full clustering breaks down due to extreme sparcity.

Other extensions include application of TURN* to categorical data, modifying TURN* to accommodate physical constraints in the case of spatial data

including constraints which can be queried (height of the wall, length of the bridge, etc.), parallel implementation of the algorithms and so forth.

An interesting application is using TURN* to detect outliers. It allows for strong outlier detection significantly improving on the work currently available which tends to include boundary points as outliers and is also relatively more dependent on parameter settings.

It is also proposed to explore synthetic data set generation in many dimensions for testing the algorithms. This is challenging because these data sets need to contain noise while ensuring that the specified number of clusters is retained. Noise should not break clusters or create them as it will typically do if applied randomly (and thus with no guarantee of uniformity). It will also be interesting to create an interface for ad hoc visual interaction with TURN* and FuzzyTURN so the clustering structure of the data can be visually explored.

# Bibliography


[1] C.C. Aggarwal and P.S. Yu. Redefining clustering for high-dimensional applications. *IEEE Transactions on Knowledge and Data Engineering*, 2002.

[2] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 94–105, 1998.

[3] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'90)*, pages 322–331, 1990.

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *Proc. Int'l Conf. Database Theory*, 1999.

[5] T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452, 1993.

[6] Silverman B.W. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 1986.

[7] Ming-Syan Chen, Jong Soo Park, and Philip S. Yu. Efficient data mining for path traversal patterns. *IEEE Transaction on Knowledge and Data Engineering*, 10(2):209–221, March/April 1998.

[8] Scott D.W. *Multivariate Density Estimation.* John Wiley and Sons, 1992.

[9] C. Eldershaw and M. Hegland. Cluster analysis using triangulation. In B.J. Noye and M.D. Teuber, editors, *Computational techniques and applications*. World Scientific, 1997.

[10] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.

[11] V. Estivill-Castro and M.E. Houle. Clustering of large geo-referenced data sets. In *Proc. 3rd Pacific Asia Conference on Knowledge Discovery and Data Mining*, pages 327–337, 1999.

[12] V. Estivill-Castro and I. Lee. Autoclust+: Automatic clustering of point-data sets in the presence of obstacles. In *Proc. International Workshop on Spatal, Temporal and Spatio-Temporal Data Mining*, pages 131 – 144, 2000.

[13] V. Estivill-Castro and I. Lee. Autoclust: Automatic clustering via boundary extraction for mining massive point-data sets. In *Proc. 5th International Conference on Geocomputation*, 2000.

[14] C. Faloutsos and K. Lin. Fastmap: a fast algorithm for indexing, datamining and visualisation of traditional and multimedia datasets. In *Proc. of the ACM SIGMOD Conference (SIGMOD'95)*, 1995.

[15] Andrew Foss, Weinan Wang, and Osmar R. Zaïane. A non-parametric approach to web log analysis. In *Proc. of Workshop on Web Mining in First International SIAM Conference on Data Mining (SDM2001)*, pages 41–50, Chicago, April 2001.

[16] Yongjian Fu, Kanwalpreet Sandhu, and Ming-Yi Shih. Clustering of web users based on access patterns. *Workshop on Web Usage Analysis and User Profiling (WEBKDD99)*, August 1999.

[17] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. Of SIGMOD'98*, Seattle, Washington, 1998.

[18] Studipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: a robust clustering algorithm for categorical attributes. In *15th Int'l Conf. on Data Eng.*, 1999.

[19] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3), December 2001.

[20] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: an attribute-oriented approach. In *18th Int'l Conf. on Very Large Databases*, pages 547–559, vancouver, canada, august 1992.

[21] Jiawei Han and Micheline Kamber. *Data Mining, Concepts and Techniques.* Morgan Kaufmann, 2001.

[22] Jiawei Han, Micheline Kamber, and Anthony K.H. Tung. Spatial clustering methods in data mining: A survey. In H. Miller and J. Han, editors, *Geographic Data Mining and Knowledge Discovery.* Taylor and Francis, 2001.

[23] D. Harel and Y. Koren. Clustering spatial data using random walks. Technical report, Weizmann Institute of Science, Rehovot, Israel, 2001.

[24] David Harel and Yehuda Koren. Clustering spatial data using random walks. In *Proceedings of KDD-2001*, 2001.

[25] S. Hauck and G. Borriello. An evaluation of bipartitioning technique. In *Proc. Chapel Hill Conference on Advanced Research in VLSI*, 1995.

[26] Bruce Hendrickson and Robert Leland. Amultilevel algorithm for partitioning graphs. Technical report, Sandia National Laboratories, 1993. citeseer.nj.nec.com/karypis97multilevel.html.

[27] A. Hinneburg and D.A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proc. 4rd Int. Conf. on Knowledge Discovery and Data Mining*, 1998.

[28] A. Hinneburg and D.A. Keim. Optimal grid-clustering: Toward breaking the curse of dimensionality in high-dimensional clustering. In *Proc. 25th VLDB Conf.*, 1999.

[29] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2:283–304, 1998.

[30] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[31] J.MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Math. Statist. Prob.*, 1967.

[32] A. Joshi and K. Joshi. On mining web access logs. Technical report, CSEE Department, UMBC, 1999. http://www.csee.umbc.edu/~ajoshi/web-mine/tr1.ps.gz.

[33] I. Kang, T. Kim, and K. Li. A spatial data mining method using Delaunay Triangulation. In *Proc. 5th international workshoop on advances in geographic information sysytems (GIS-97)*, pages 35–39, 1997.

[34] K.V.R Kanth, D. Agarwal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD Conf.*, 1998.

[35] G. Karypis and V. Kumar. Metis: Unstructured graph partitioning and sparse matrix ordering system. Technical report, CS Dept., University of Minnesota, 1995.

[36] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Hypergraph partitioning: Applications in vlsi domain. Tech-

nical report, CS Dept., University of Minnesota, 1996. citeseer.nj.nec.com/karypis97multilevel.html.

[37] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. Technical report, CS Dept., University of Minnesota, 1997. citeseer.nj.nec.com/karypis97multilevel.html.

[38] George Karypis, Eui-Hong Han, and Vipin Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8):68–75, August 1999.

[39] T. Kohenen. *Self-Organizing Maps.* Springer-Verlag, 1995.

[40] L.Kaufman and P.J.Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[41] Spiliopoulou M. and Pohle C. Data mining for measuring and improving the success of web sites. *Data Mining and Knowledge Discovery, Special Issue on Electronic Commerce*, 2000.

[42] M.Ankerst, M.Breunig, H.-P. Kriegel, and J.Sander. Optics: Ordering points to identify the clustering structure. In *Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99)*, pages 49–60, 1999.

[43] Timothy Masters. *Neural, Novel & Hybrid Algorithms for Time Series Prediction.* John Wiley and Sons, 1995.

[44] Bruce Matichuk and Osmar R. Zaïane. Unsupervised classification of sound for multimedia indexing. In *First Intl. Workshop on Multimedia Data Mining (MDM/KDD'2000)*, pages 31–36, Boston, MA, August 2000.

[45] B. Mobasher, R. Cooly, and J. Srivastava. Automatic personalization based on web usage mining. Technical Report TR99-010, Department of Computer Science, Depaul University, 1999.

[46] B. Mobasher, N. Jain, E. Han, and J. Srivastava. Web mining: pattern discovery from world wide web transactions. Technical report, Department of Computer Science, University of Minnesota, 1996.

[47] J. Moore, E. Han, D. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, and B. Mobasher. Web page categorization and feature selection using association rule and principal component clustering. In *Proc. 7th Workshop on Information Technologies and Systems*, 1997.

[48] M.D. Mulvenna, S. S. Anand, and A. G. Büchner. Personalization on the net using web mining. *Communications of the ACM*, 43(8), August 2000.

[49] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. On Very Large Data Bases (VLDB'94)*, pages 144–155, Santiago, Chile, September 1994.

[50] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Intl. Conf. on Very Large Data Bases (VLDB'94)*, pages 144–155, Santiago, Chile, September 1994.

[51] G. Salton and M.J. McGill. *Introduction to modern information retrieval.* McGraw-Hill, New York, 1982.

[52] J.W. Sammon. A non-linear mapping for data structure analysis. *IEEE Transactions on Computers*, 2002.

[53] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery, An International Journal*, 1998.

[54] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: a multi-resolution clustering approach for very large spatial databases. In *24th VLDB Conference*, New York, USA, 1998.

[55] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. A wavelet-based clustering approach for spatial data in very large databases.

*The International Journal on Very Large Databases*, 8(4):289–304, February 2000.

[56] P. Smyth. Clustering using monte-carlo cross-validation. In *ACM Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'96)*, 1996.

[57] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations*, Jan 2000.

[58] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000.

[59] A. Thomasian, V. Castelli, and C.-S. Li. Clustering and singular value decomposition for approximate indexing in high dimensional spaces. In *Proc. Conf. Information and Knowledge Management*, 1998.

[60] J. Tsai, M. Gerstain, and M. Levitt. Simulating the minimum core for hydrophobic collapse in globular proteins. *Protein Science*, 1997.

[61] W. Wang, J. Yang, and R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proc. 1997 Int. Conf. on Very Large Data Bases (VLDB'97)*, pages 186–195, 1997.

[62] Osmar R. Zaïane, Andrew Foss, Chi-Hoon Lee, and Weinan Wang. On data clustering analysis: Scalability, constraints and validation. In *Proc. of the Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'02)*, 2002.

[63] Osmar R. Zaïane, Man Xin, and Jiawei Han. Discovering web access patterns and trends by applying OLAP and data mining technology on web logs. In *Proc. Advances in Digital Libraries ADL'98*, pages 19–29, Santa Barbara, CA, USA, April 1998.

[64] Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In *Proc. ACM SIGIR'98*, 1998.

[65] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *1996 ACM SIGKDD Int. Conf. Managament of Data*, pages 103–114, June 1996.